

# MISTA 2005

Proceedings of the

## **2<sup>nd</sup> Multidisciplinary International Conference on Scheduling: Theory and Applications**

18<sup>th</sup> – 21<sup>st</sup> July 2005  
New York, USA

*Edited by*

**Graham Kendall**, University of Nottingham, UK

**Lei Lei**, Rutgers University, USA

**Michael Pinedo**, New York University, USA

Published by Stern School of Business, New York University, USA

ISSN 2305-249X

ISBN-10: 0-9545821-4-4

ISBN-13: 978-0-9545821-4-2

## **Editors**

### **Graham Kendall**

School of Computer Science and Information Technology  
University of Nottingham  
Jubilee Campus  
Wollaton Road  
Nottingham  
NG8 1BB  
UK  
Email: gxk@cs.nott.ac.uk

### **Lei Lei**

Department of Management Sciences & Information Systems  
Rutgers Business School – Newark and New Brunswick Campuses  
Rutgers University  
180 University Avenue  
Newark, NJ 07102  
E-mail: llei@business.rutgers.edu

### **Michael Pinedo**

Dept. of Information, Operations and Management Sciences  
Stern School of Business  
44 West Fourth Street  
New York University  
New York, NY 10012  
E-mail: mpinedo@stern.nyu.edu

## MISTA 2005 Conference Program Committee

|                                  |                                                                         |
|----------------------------------|-------------------------------------------------------------------------|
| <b>Graham Kendall</b> (co-chair) | The University of Nottingham, UK                                        |
| <b>Lei Lei</b> (co-chair)        | Rutgers University, USA                                                 |
| <b>Mike Pinedo</b> (co-chair)    | New York University, USA                                                |
| Belarmino Adenso-Diaz            | Universidad de Oviedo, Spain                                            |
| Samad Ahmadi                     | De Montfort University, UK                                              |
| Uwe Aickelin                     | The University of Nottingham, UK                                        |
| Hesham Alfares                   | King Fahd University of Petroleum and Minerals,<br>Saudi Arabia         |
| Ali Allahverdi                   | Kuwait University, Kuwait                                               |
| Abdelhakim Artiba                | Facultes Universitaires Catholiques de Mons (CREGI<br>- FUCAM), Belgium |
| Philippe Baptiste                | CNRS, Ecole Polytechnique, France                                       |
| James Bean                       | University of Oregon, USA                                               |
| J. Christopher Beck              | University of Toronto, Canada                                           |
| Michael Bender                   | SUNY Stony Brook, USA                                                   |
| Jacek Blazewicz                  | Poznan University of Technology, Poland                                 |
| Peter Brucker                    | University of Osnabrueck, Germany                                       |
| Edmund Burke                     | The University of Nottingham, UK                                        |
| Xiaoqiang Cai                    | The Chinese University of Hong Kong, Hong Kong                          |
| Jacques Carlier                  | Université de Technologie de Compiègne, France                          |
| Patrick De Causmaecker           | KaHo St.-Lieven, Belgium                                                |
| Zhi-Long Chen                    | University of Maryland, USA                                             |
| T. C. Edwin Cheng                | The Hong Kong Polytechnic University, Hong Kong                         |
| Philippe Chretienne              | Paris 6 University, LIP6 laboratory, France                             |
| Ed Coffman                       | Columbia University, USA                                                |
| Mauro Dell'Amico                 | University of Modena and Reggio Emilia, Italy                           |
| Erik Demeulemeester              | Katholieke Universiteit Leuven, Belgium                                 |
| Kath Dowsland                    | Gower Optimal Algorithms Ltd., UK                                       |
| Moshe Dror                       | University of Arizona, USA                                              |
| Maciej Drozdowski                | Poznan University of Technology, Poland                                 |
| Alejandra Duenas                 | Coventry University, UK                                                 |
| Janet Efstathiou                 | University of Oxford, UK                                                |
| Wilhelm Erben                    | Univ of Applied Scs Konstanz, Germany                                   |
| Dror Feitelson                   | The Hebrew University, Israel                                           |
| Gerd Finke                       | Laboratory LEIBNIZ-IMAG, France                                         |
| Peter Fleming                    | University of Sheffield, UK                                             |
| David Fogel                      | Natural Selection, USA                                                  |
| Dalibor Froncek                  | University of Minnesota, USA                                            |
| Michel Gendreau                  | Université de Montréal, Canada                                          |
| Celia Glass                      | City University, UK                                                     |
| Fred Glover                      | University of Colorado, USA                                             |
| Bernard Grabot                   | Laboratoire Génie de Production, ENIT, France                           |
| Alain Guinet                     | INSA de Lyon, France                                                    |
| Nicholas Hall                    | Ohio State, USA                                                         |
| Jin-Kao Hao                      | University of Angers, France                                            |
| Martin Henz                      | National University of Singapore, Singapore                             |
| Jeffrey Herrmann                 | University of Maryland, USA                                             |
| Willy Herroelen                  | Katholieke Universiteit Leuven, Belgium                                 |

Han Hoogeveen  
 Toshihide Ibaraki  
 Adam Janiak  
 Jeffrey H. Kingston  
 Wieslaw Kubiak  
 Raymond Kwan  
 Claude Le Pape  
 Chung-Yee Lee  
  
 Joseph Leung  
 Eugene Levner  
 Arne Løkketangen  
 Irv Lustig  
 Dirk Mattfeld  
 Martin Middendorf  
 Rolf Möhring  
 David Montana  
 Alix Munier  
 Alexander Nareyek  
 Bryan A. Norman  
 Wim Nuijten  
 Ibrahim H. Osman  
 Costas Pappis  
 Erwin Pesch  
 Dobrila Petrovic  
 Sanja Petrovic  
 Chris Potts  
 Jean-Yves Potvin  
 Christian Prins  
 Kirk Pruhs  
 Vic Rayward-Smith  
 Colin Reeves  
 Celso Ribeiro  
 Andrea Schaerf  
 Andreas S. Schulz  
 Roman Slowinski  
 Stephen F. Smith  
 Chelliah Sriskandarajah  
 Roberto Tadei  
 Jonathan Thompson  
 Vincent T'Kindt  
 Michael Trick  
 Denis Trystram  
 Edward Tsang  
 Greet Vanden Berghe  
 Alkis Vazacopoulos  
 Steef van de Velde  
 Stefan Voss  
 Jean-Paul Watson  
 Joel Wein  
  
 George White  
 Darrell Whitley

Utrecht University, The Netherlands  
 Kwansei Gakuin University, Japan  
 Wroclaw University of Technology, Poland  
 University of Sydney, Australia  
 MUN, St. John's, Canada  
 University of Leeds, UK  
 ILOG, France  
 The Hong Kong University of Science and  
 Technology, Hong Kong  
 New Jersey Institute of Technology, USA  
 Holon Institute of Technology, Israel  
 Molde College, Norway  
 ILOG, USA  
 Technical University Braunschweig, Germany  
 University of Leipzig, Germany  
 Technische Universität Berlin, Germany  
 BBN Technologies, USA  
 LIP6, University Paris 12, Paris  
 Cork Constraint Computation Centre, Ireland  
 University of Pittsburgh, USA  
 ILOG, France  
 American University of Beirut, Lebanon  
 University of Piraeus, Greece  
 University of Siegen, Germany  
 Coventry University, UK  
 The University of Nottingham, UK  
 University of Southampton, UK  
 Université de Montreal, Canada  
 UTT, France  
 University of Pittsburgh, USA  
 University of East Anglia, UK  
 Coventry University, UK  
 Catholic University of Rio de Janeiro, Brazil  
 University of Udine, Italy  
 Massachusetts Institute of Technology, USA  
 Poznan University of Technology, Poland  
 Carnegie Mellon University, USA  
 The University of Texas at Dallas, USA  
 Politecnico di Torino, Italy  
 Cardiff University, UK  
 Polytech 'Tours, University of Tours, France  
 Carnegie Mellon University, USA  
 ID - IMAG, France  
 University of Essex, UK  
 KaHo St.-Lieven, Belgium  
 Dash Optimization, USA  
 RSM Erasmus University, Netherlands  
 University of Hamburg, Germany  
 Sandia National Laboratories, USA  
 Polytechnic University, Brooklyn NY, and Akamai  
 Technologies, USA  
 University of Ottawa, Canada  
 Colorado State University, USA



|                   |                                             |
|-------------------|---------------------------------------------|
| Gerhard Woeginger | TU Eindhoven, The Netherlands               |
| Qingfu Zhang      | University of Essex, UK                     |
| Yakov Zinder      | University of Technology, Sydney, Australia |

## **MISTA 2005 Organizing Committee**

|                          |                                               |
|--------------------------|-----------------------------------------------|
| <b>Jian Yang</b> (chair) | New Jersey Institute of Technology, USA       |
| Layek Abdel-Malek        | New Jersey Institute of Technology, USA       |
| Athanassios Bladikas     | New Jersey Institute of Technology, USA       |
| Sanchoy Das              | New Jersey Institute of Technology, USA       |
| Shuguang Liu             | State University of New York (New Palze), USA |
| Vijay B Reddy Onteddu    | New Jersey Institute of Technology, USA       |
| Alex Root                | New York University                           |
| Corinne Schiavo          | Rutgers University, USA                       |
| Shitao Yang              | Seton Hall University                         |
| Xiaolong Zhang           | Fairleigh Dickinson University                |
| Yao Zhao                 | Rutgers University, USA                       |
| Hua Zhong                | Rutgers University, USA                       |

## **MISTA 2005 International Advisory Committee**

|                               |                                                                               |
|-------------------------------|-------------------------------------------------------------------------------|
| <b>Graham Kendall</b> (chair) | The University of Nottingham, UK                                              |
| Abdelhakim Artiba             | Facultes Universitaires Catholiques de Mons (CREGI - FUCAM), Belguim          |
| James Bean                    | University of Michigan, USA                                                   |
| Jacek Blazewicz               | Institute of Computing Science, Poznan University of Technology, Poland       |
| Peter Brucker                 | University of Osnabrueck, 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                                                   |
| Fred Glover                   | Leeds School of Business, University of Colorado, USA                         |
| Bernard Grabot                | Laboratoire Génie de Production - Equipe Production Automatisée, France       |
| Claude Le Pape                | ILOG, France                                                                  |
| Toshihide Ibaraki             | Kyoto University, Japan                                                       |
| Michael Pinedo                | New York University, USA                                                      |
| Ibrahim Osman                 | American University of Beirut, Lebanon                                        |
| Jean-Yves Potvin              | Université de Montreal, 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 very many people.

The editors would like to thank the organizing committee who have done an exceptional job in making all the necessary arrangements that have allowed this conference to take place. Special appreciation is extended to Jian Yang, who chaired the organising committee. He has worked extremely hard in order for this conference to be a success.

We would like to thank our web master, Vijay Onteddu, for providing the necessary support in maintaining both the web pages and the on-line review system.

A special thank you goes to Yihong Fan, for her tremendous amount of effort in helping to prepare the conference proceedings.

Thank you to the program committee for doing such an excellent job of reviewing the papers.

The editors would also like to thank the international advisory committee for all their helpful advice and comments during the planning and preparation of this conference.

Several sponsors have provided financial support for the conference. We appreciate the support from AmerisourceBergen, Dash Optimization, New York University (Stern School of Business), Rutgers University (Rutgers Business School), and The National Science Foundation.

The editors would like to extend special thanks to Corinne Schiavo. She was a constant source of help and advice and her professionalism has made the task of organising this conference a lot easier than it would otherwise have been.

# Contents

## Plenary Speakers

|                                                                              |   |
|------------------------------------------------------------------------------|---|
| Good Neighborhoods and Well-Matched Neighborhoods<br><i>Balas E.</i> .....   | 2 |
| Seamless Download Scheduling in Computer Networks<br><i>Coffman E.</i> ..... | 3 |
| Scheduling, a Critical Biography<br><i>Lenstra J. K.</i> .....               | 4 |

## Tutorial Speakers

|                                                                                              |   |
|----------------------------------------------------------------------------------------------|---|
| Shop Scheduling with Blocking<br><i>Brucker P.</i> .....                                     | 6 |
| Stochastic Scheduling Subject to Preemptive-Repeat Machine Breakdowns<br><i>Cai X.</i> ..... | 6 |
| Hybrid Technologies for Supply Chain Planning and Scheduling<br><i>Vazacopoulos A.</i> ..... | 6 |
| Constraint Satisfaction<br><i>Tsang E.</i> .....                                             | 6 |
| Supply Chain Scheduling<br><i>Hall N.</i> .....                                              | 6 |
| Scheduling in Uncertain and Dynamic Environments<br><i>Smith S.</i> .....                    | 6 |

## Alternative Formulations: Abstracts

|                                                                                                    |    |
|----------------------------------------------------------------------------------------------------|----|
| A TSP Formulation of the DNA Sequencing Problem<br><i>Nikolakopoulos A. and Sarimveis H.</i> ..... | 8  |
| Multi-Criteria Soft Constraints in Timetabling<br><i>Rudova H. and Vlk M.</i> .....                | 11 |
| Models for Adjacent Resource Scheduling<br><i>Duin C.W. and van der Sluis H.J.</i> .....           | 16 |

## Applications: Papers

|                                                                                                                                                         |    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Scheduling Multiple Types of Time-Shared Aircraft: Models and Methods for Practice<br><i>Karaesmen I. Z., Keskinocak P., Tayur S. and Yang W.</i> ..... | 19 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|----|

## Applications: Abstracts

|                                                                                                                                                                       |    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Systems Design for Scheduling—Open Tools<br><i>De Causmaecker P., Demeester P., VandenBerghe G. and Verbeke B.</i> .....                                              | 40 |
| A Deterministic Scheduling Approach for Wafer Fabrication Facilities<br><i>Mason S. J., Pfund M. and Fowler J.</i> .....                                              | 42 |
| Multiple Orders Per Job Scheduling Problems<br><i>Fowler J., Kutanoglu E. and Mason S.</i> .....                                                                      | 44 |
| Optimal Reactive Scheduling for Job Shop, Make-to-order Industries: A Mould Making Industry Case Study<br><i>Gomes M. C., Barbosa-Povoa A. and Novais A. Q.</i> ..... | 46 |

## **Batch Scheduling: Papers**

|                                                                                                                                  |    |
|----------------------------------------------------------------------------------------------------------------------------------|----|
| Multiple Orders Per Job Batch Scheduling on a Single Machine with Incompatible Jobs<br><i>Erramilli V. and Mason S. J.</i> ..... | 48 |
|----------------------------------------------------------------------------------------------------------------------------------|----|

## **Batch Scheduling: Abstracts**

|                                                                                                           |    |
|-----------------------------------------------------------------------------------------------------------|----|
| A Decomposition Approach for Batch Scheduling in the Processing Industry<br><i>Maravelias C. T.</i> ..... | 66 |
|-----------------------------------------------------------------------------------------------------------|----|

## **Delivery Scheduling: Abstracts**

|                                                                                                                          |    |
|--------------------------------------------------------------------------------------------------------------------------|----|
| How to Maximize Rewards in an Incentived Made-to-order Scheduling Environment<br><i>Glass C. A. and Ng D. S.T.</i> ..... | 69 |
| On the Capacitated Packing-Delivering Problem with a Fixed Route<br><i>Lei L., Park S. and Liu S.</i> .....              | 70 |

## **Evolutionary Algorithms: Abstracts**

|                                                                                                                                                                         |    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Ant Colony Optimisation for Power Plant Maintenance Scheduling Optimization: A Simplified Hydro Tasmania System<br><i>Foong W., Maier H. R. and Simpson A. R.</i> ..... | 72 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|

## **Job-shop Scheduling: Papers**

|                                                                                                                                     |    |
|-------------------------------------------------------------------------------------------------------------------------------------|----|
| An Approach to Predictive-reactive Scheduling of Parallel Machines Subject to Disruptions<br><i>Duenas A. and Petrovic D.</i> ..... | 74 |
|-------------------------------------------------------------------------------------------------------------------------------------|----|

## **Job-shop Scheduling: Abstracts**

|                                                                                                                                  |    |
|----------------------------------------------------------------------------------------------------------------------------------|----|
| Approximate Solutions for a New Version of the Car Sequencing Problem<br><i>Adenso-Diaz B., Bautista J. and Pereira J.</i> ..... | 93 |
|----------------------------------------------------------------------------------------------------------------------------------|----|

## **Local Search: Papers**

|                                                                                                                                          |    |
|------------------------------------------------------------------------------------------------------------------------------------------|----|
| Random Walks and Neighbourhood Bias in Oversubscribed Scheduling<br><i>Roberts M., Whitley L. D., Howe A. E. and Barbulescu L.</i> ..... | 98 |
|------------------------------------------------------------------------------------------------------------------------------------------|----|

## **Local Search: Abstracts**

|                                                                                                                         |     |
|-------------------------------------------------------------------------------------------------------------------------|-----|
| Tabu Search Algorithms For Cyclic Machine Scheduling Problems with Blocking<br><i>Brucker P. and Kampmeyer T.</i> ..... | 107 |
| Repair-cost-guided Local Search for Scheduling<br><i>Laplagne I., Kwan R. S. K. and Kwan A. S. K.</i> .....             | 109 |

## **Machine Scheduling: Papers**

|                                                                                                                                                        |     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Bicriteria Scheduling of Equal Length Jobs with Release Dates on Identical Parallel Machines<br><i>Balasubramanian H., Fowler J. and Keha A.</i> ..... | 112 |
| Enumeration of Pareto Optima for Uniform Parallel Machine Scheduling with Two Criteria<br><i>Bouibede-Hocine K. and T'kindt V.</i> .....               | 123 |
| Heuristic Approaches to a Common Due Date Earliness-tardiness Scheduling Problem<br><i>Gordon V. S. and Tarasevich A. A.</i> .....                     | 132 |
| The Single Machine Just-in-time Scheduling Problem with Preemptions<br><i>Hendel Y. and Sourd F.</i> .....                                             | 140 |

|                                                                                                                                                                      |     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| The Single-machine Earliness-tardiness Scheduling Problem with Due Date Assignment and Resource-dependent Processing Times<br><i>Steiner G. and Shabtay D.</i> ..... | 149 |
| A Branch-and-bound Algorithm for Single Machine Total Tardiness Problem with Release Dates<br><i>Oguz C. and Yucel E.</i> .....                                      | 162 |
| Large Neighborhood for a Parallel Scheduling Problem with Earliness-tardiness Penalties and a Common Due Date<br><i>Rios Solis Y. A. and Sourd F.</i> .....          | 169 |

### **Machine Scheduling: Abstracts**

|                                                                                                                                                             |     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| A Divide-and-merge Algorithm for Single Machine Weighted Tardiness Problem<br><i>Keha A. and Fowler J.</i> .....                                            | 186 |
| Job Shop Scheduling with Earliness, Tardiness and Intermediate Inventory Holding Costs<br><i>Bulbul K.</i> .....                                            | 188 |
| Scheduling a Single Machine to Minimize a Regular Objective Function under Setup Constraints<br><i>Pape C. L. and Baptiste P.</i> .....                     | 193 |
| Efficient Algorithms for Machine Scheduling Problems with Earliness and Tardiness Penalties<br><i>Feng G. and Lau H. C.</i> .....                           | 196 |
| Stable Single-machine Schedules<br><i>Leus R. and Herroelen W.</i> .....                                                                                    | 213 |
| Lower Bounds for Parallel Machine Scheduling<br><i>Savourey D., Jouglet A. and Baptiste P.</i> .....                                                        | 215 |
| Job Shop Scheduling Problem with Multi-purpose Machines and Availability Constraints<br><i>Zribi N., Kamel A. E. and Borne P.</i> .....                     | 217 |
| Complexity Results for the Parallel Machine Scheduling Problems with a Server in Computer Systems<br><i>Guirchoun S., Souhkal A. and Martineau P.</i> ..... | 232 |

### **Multi-processor Scheduling: Papers**

|                                                                                                                                                                |     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| A Service Level Agreement Protocol and Tabu Search for Scheduling on Computational Grids<br><i>Ouelhadj D., Garibaldi J.M. and Sakellariou R.</i> .....        | 238 |
| How Good are SPT Schedules for Fair Optimality Criteria<br><i>Angel E., Bampis E. and Pascual F.</i> .....                                                     | 244 |
| A Heuristic Algorithm for Scheduling ‘Blocked Out’ Units in Continuous Processing Industry: Preliminary Results<br><i>Bose S. K. and Bhattacharya S.</i> ..... | 258 |

### **Production Scheduling: Papers**

|                                                                                                                                                                      |     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| A Bi-criteria Approach to Scheduling Wafer Fabrication Facilities<br><i>Pfund M., Balasubramanian H., Fowler J. and Mason S.</i> .....                               | 276 |
| Minimizing Makespan with Multiple Orders Per Job in a Two Machine Flowshop with Item Processing<br><i>Laub J. D., Fowler J. W., and Keha A.</i> .....                | 288 |
| A Sequential Pair Exchange Approach for Solving Single-machine Scheduling Problems with General Earliness-tardiness Penalties<br><i>Lin S-W. and Chou S-Y.</i> ..... | 299 |

## Production Scheduling: Abstracts

|                                                                                                                                                                              |     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Lower and Upper Bounds for the Job-shop Scheduling Problem with Sequence-dependent Setup Times<br><i>Artigues C., Buscaylet F. and Feillet D.</i> .....                      | 316 |
| Proactive Resource Allocation Heuristics for Robust Project Scheduling<br><i>Braeckmans K., Demeulemeester E. and Herroelen W.</i> .....                                     | 322 |
| Optimal Production Scheduling for Dairy Industries<br><i>Doganis P., Sarimveis H., Alexandridis A. and Patrinos P.</i> .....                                                 | 324 |
| Approximations to Optimal $k$ -unit Cycles for Dual Gipper Robot Cells<br><i>Geismar H. N., Dawande M. and Sriskandaranjah C.</i> .....                                      | 326 |
| The Unit-Operation-Stock Superstructure (UOSS) and the Quality-Logic-Quality Paradigm (QLQP) for Production Scheduling in the Process Industries<br><i>Kelly J. D.</i> ..... | 327 |
| Forming and Scheduling Jobs with Order Weights in Semiconductor Manufacturing<br><i>Tanrisever F. and Kutanoglu E.</i> .....                                                 | 334 |
| A Tabu-Search Based Reactive Procedure for the RCPSP with Stochastic Resource Availabilities<br><i>Lambrechts O., Demeulemeester E. and Herroelen W.</i> .....               | 335 |
| A Survey on Minimizing Makespan for the Preemptive job shop with Two Machines<br><i>Kimbrel T.</i> .....                                                                     | 337 |
| Single Machine Product Scheduling with Outsourcing Allowed<br><i>Sung C. S., Lee I. S., Yoon S. H. and Juhn J.H.</i> .....                                                   | 340 |
| An Evaluation of Exact and Suboptimal Procedures for Constructing Stable Project Schedules<br><i>Van de Vonder S., Demeulemeester E. and Herroelen W.</i> .....              | 341 |

## Real-time Scheduling: Papers

|                                                                                                                      |     |
|----------------------------------------------------------------------------------------------------------------------|-----|
| Periodic Real-time Scheduling: From Latency-based Model to Deadline-based Model<br><i>Cucu L. and Sorel Y.</i> ..... | 344 |
| A Comparison of Combinatorial Optimization and Dispatch Rules for Online Scheduling<br><i>Montana D.</i> .....       | 353 |

## Real World Scheduling: Papers

|                                                                                                                                               |     |
|-----------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Job Shop Scheduling with Lot-sizing and Batching in an Uncertain Real-world Environment<br><i>Petrovic S., Fayad C. and Petrovic D.</i> ..... | 363 |
| A History of Decision-Making Tools for Production Scheduling<br><i>Herrmann J. W.</i> .....                                                   | 380 |

## Real World Scheduling: Abstracts

|                                                                                                                                                                                                                                                                                                                         |     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Challenges in Matching Assets with Demand across All Supply Chain Management Decision Tiers in the IBM Microelectronics Division from Automation Fabs to Central Planning Engines to Collaborative Behavior (Work Around)<br><i>Fordyce K., Burda R., Degbotse A., Milne J., Morrison J. R. and Williamson E.</i> ..... | 391 |
| Scheduling Wafer Fabrication Operations: State-of-the-practice and Future Needs<br><i>Fowler J. and Pfund M.</i> .....                                                                                                                                                                                                  | 392 |
| Scheduling Product Families in a High Volume, Flexible, Assemble-to-order Factory<br><i>Monkman S. K., Morrice D. J. and Bard J. F.</i> .....                                                                                                                                                                           | 394 |

## **Rostering: Papers**

|                                                                                                                                        |     |
|----------------------------------------------------------------------------------------------------------------------------------------|-----|
| A Decomposition, Construction and Post-processing Approach for Nurse Rostering<br><i>Brucker P., Qu R., Burke E. and Post G.</i> ..... | 397 |
|----------------------------------------------------------------------------------------------------------------------------------------|-----|

## **Rostering: Abstracts**

|                                                                                                                 |     |
|-----------------------------------------------------------------------------------------------------------------|-----|
| Integrating Nurse and Surgery Scheduling<br><i>Beliën J. and Demeulemeester E.</i> .....                        | 408 |
| An Electromagnetic Meta-heuristic for the Nurse Scheduling Problem<br><i>Maenhout B. and Vanhoucke M.</i> ..... | 410 |

## **Search Algorithms: Papers**

|                                                                                                                                                     |     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| An Investigation of Variable Neighbourhood Search for the University Course<br>Timetabling<br><i>Abdullah S., Burke E. K. and McCollum B.</i> ..... | 413 |
| A Zero-Inventory Production-Distribution Problem with Time-Window Constraints<br><i>Armstrong R., Lei L. and Gao S.</i> .....                       | 428 |
| Scheduling UET-UCT Task Systems under the Out-forest Precedence Constraints<br><i>Hanen C. and Zinder Y.</i> .....                                  | 445 |
| The Worst-case Analysis of the Garey-Johnson Algorithm for Preemptive Tasks on $m$<br>Processors<br><i>Hanen C. and Zinder Y.</i> .....             | 453 |
| Scheduling Orders in Dedicated and Flexible Machine Environments<br><i>Leung J. Y. T., Li H. and Pinedo M.</i> .....                                | 471 |
| Retractable Contract Network for Distributed Scheduling<br><i>Tsang E. P. K., Gosling T., Virginas B., Voudouris C. and Owusu G.</i> .....          | 485 |
| Scheduling Algorithms for Master-slave Systems<br><i>Leung J. Y. T. and Zhao H.</i> .....                                                           | 501 |

## **Search Algorithms: Abstracts**

|                                                                                                                                                     |     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Skills-Based Agent Scheduling at Call Centers<br><i>Aykin T. and McKenna J. S.</i> .....                                                            | 515 |
| Timetable Synchronization for Mass Transit<br><i>Leung J. M. Y. and Wong R. C. W.</i> .....                                                         | 516 |
| Algorithms for Online Buffer Management<br><i>Mahdian M., Bansal N., Fleischer L. K., Kimbrel T., Schieber B. and Sviridenko M.</i> .....           | 518 |
| A Practitioner's Approach to Distribution and Supply Chain Planning and Management<br><i>Miller T.</i> .....                                        | 519 |
| Job Shop Scheduling with Set-up Times, Deadlines and Precedence Constraints<br><i>Balas E., Simonetti N. and Vazacopoulos A.</i> .....              | 520 |
| Worst-case Analysis for Queueing Service Policies<br><i>Wang Z. and Xing W.</i> .....                                                               | 533 |
| Integrated Scheduling of Different Types of Handling Equipment at Automated<br>Container Terminals<br><i>Lau H. Y. K., Zhao Y. and Pan C.</i> ..... | 536 |

## **Supply Chain Scheduling: Papers**

|                                                                                                                                    |     |
|------------------------------------------------------------------------------------------------------------------------------------|-----|
| Single Machine Multiple Orders Per Job Scheduling Using Column Generation<br><i>Jampani J., Mason S. J. and Erramilli V.</i> ..... | 539 |
|------------------------------------------------------------------------------------------------------------------------------------|-----|

## Supply Chain Scheduling: Abstracts

|                                                                                                                                                                   |     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Integrated Production-distribution Scheduling to Optimize the Tradeoff between Order Delivery Performance and Total Cost<br><i>Pundoor G. and Chen Z-L.</i> ..... | 549 |
| Combined Production and Distribution Scheduling of Ready-mix Concrete<br><i>Fong P. Y. and Piplani R.</i> .....                                                   | 551 |
| Incremental Propagation Rules for a Precedence Graph with Optional Activities and Time Windows<br><i>Bartak R. and Cepek O.</i> .....                             | 552 |

## Theoretical Scheduling: Papers

|                                                                                                                                                                             |     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Scheduling on a Single Machine to Minimize Total Flow Time with Job Rejections<br><i>Bunde D. P.</i> .....                                                                  | 562 |
| Online Scheduling of Precedence Constrained Tasks<br><i>Huo Y. and Leung J. Y-T.</i> .....                                                                                  | 573 |
| Bicriteria Parallel Task Scheduling<br><i>Baille F., Bampis E., Laforest C. and Rapine C.</i> .....                                                                         | 585 |
| On Project Scheduling with And/or Precedence Constraints<br><i>Adelson-Velsky G., Gelbukh A. and Levner E.</i> .....                                                        | 597 |
| New Tight NP-hardness of Preemptive Multiprocessor and Open Shop Scheduling<br><i>Shchepin E. and Vakhania N.</i> .....                                                     | 606 |
| Single-machine Scheduling with General Costs under Compound-type Distributions<br><i>Cai X., Wu X. and Zhou X.</i> .....                                                    | 630 |
| A Comparison of Different Formulations for the Non-preemptive Single Machine Total Weighted Tardiness Scheduling Problem<br><i>Khowala K., Keha A., and Fowler J.</i> ..... | 643 |

## Theoretical Scheduling: Abstracts

|                                                                                                                        |     |
|------------------------------------------------------------------------------------------------------------------------|-----|
| On the Complexity of Adjacent Resource Scheduling<br><i>Duin C. and van der Sluis H. J.</i> .....                      | 653 |
| Improved Approximation Algorithms for Broadcast Scheduling<br><i>Bansal N., Coppersmith D. and Sviridenko M.</i> ..... | 655 |

## Timetabling: Papers

|                                                                                         |     |
|-----------------------------------------------------------------------------------------|-----|
| Graphical Tools for Heuristic Visualization<br><i>Ranson D. and Cheng P. C-H.</i> ..... | 658 |
|-----------------------------------------------------------------------------------------|-----|

## Timetabling: Abstracts

|                                                                                                                                                          |     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Course Timetabling—Extended Tiles for University Timetabling<br><i>Adriaen M., De Causmaecker P. and VandenBerghe G.</i> .....                           | 669 |
| A Predictive-reactive Approach for RCPSP with Labour Constraints and Time-dependent Activities Requirements<br><i>Drezet L-E. and Billaut J-C.</i> ..... | 671 |
| Solving a Timetabling Problem using Bender's Decomposition<br><i>Sarin S. C., Wang Y. and Varadarajan A.</i> .....                                       | 673 |

## Transport Scheduling: Papers

|                                                                                                                                              |     |
|----------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Applying the Extended Network Simplex Algorithm to Dynamic Automated Guided Vehicles Scheduling<br><i>Rashidi H. and Tsang E. P.K.</i> ..... | 677 |
|----------------------------------------------------------------------------------------------------------------------------------------------|-----|



## **Transport Scheduling: Abstracts**

|                                                                                                                                          |     |
|------------------------------------------------------------------------------------------------------------------------------------------|-----|
| On a Certain Berth Scheduling Problem<br><i>Blazewicz J., Machowiak M., Cheng T.C. E. and Oguz C.</i> .....                              | 694 |
| Forecasting Flight Schedule Robustness<br><i>Burke E., DeCausmaecker P. , DeMaere G., Mulder J. and Paelinck M.</i> .....                | 698 |
| Scheduling the Production and Transportation Operations subject to Dynamic Arrivals<br><i>Fan C., Kumar M., Lei L. and Yang J.</i> ..... | 701 |
| Building Robustness into Train Driver Schedules<br><i>Kwan A. S.K., Laplagne I. and Kwan R. S.K.</i> .....                               | 702 |

## **Vehicle Routing: Papers**

|                                                                                                                 |     |
|-----------------------------------------------------------------------------------------------------------------|-----|
| Improved Pruning in Column Generation of a Vehicle Routing Problem<br><i>Westphal S. and Krumke S. O.</i> ..... | 705 |
|-----------------------------------------------------------------------------------------------------------------|-----|

## **Vehicle Routing: Abstracts**

|                                                                                                                         |     |
|-------------------------------------------------------------------------------------------------------------------------|-----|
| Vehicle Routing and Scheduling with Time Varying Travel Times<br><i>Eglese R. and Maden W.</i> .....                    | 724 |
| A Dynamic Programming Approach to the Truckload Routing Problem using A Triplet Formulation<br><i>Miori V. M.</i> ..... | 725 |

## **New Trends in Scheduling: Abstracts**

|                                                                                                                          |     |
|--------------------------------------------------------------------------------------------------------------------------|-----|
| Inventory Control and Scheduling with Facilities in Parallel<br><i>Krasik V., Leung J., Pinedo M. and Zhang J.</i> ..... | 740 |
|--------------------------------------------------------------------------------------------------------------------------|-----|

## **Others: Abstracts**

|                                                                                                                                                                                                           |     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Analytic Evaluation of the Expectation and Variance of Different Performance Measures for Scheduling Jobs on a Single Machine under Processing Time Variability<br><i>Sarin S. and Nagarajan B.</i> ..... | 744 |
| A Genetic Algorithm with Random Keys Approach for the Job Shop Scheduling Problem with Earliness and Tardiness<br><i>Bedoya-Valencia L., Rodriguez-Velázquez E. and Rabadi G.</i> .....                   | 745 |

## **Author Index**

## **Plenary Speakers**

# GOOD NEIGHBORHOODS AND WELL-MATCHED NEIGHBORHOODS

*Egon Balas*

Carnegie Mellon University

Scheduling procedures typically generate a starting schedule by some (possibly randomized) greedy routine, then improve it through neighborhood search. A good neighborhood is one that is (a) large and (b) easy to search. Optimizing over a good neighborhood is a promising approach. A well-matched pair of neighborhoods is one with little overlap between its members. Alternatively optimizing over the members of a well-matched pair of good neighborhoods is even more promising. These ideas will be illustrated on examples.

# Seamless Download Scheduling in Computer Networks

*Ed Coffman*  
Columbia University

Bandwidth and storage exist in computer networks in such abundance that the classical constraints of scheduling and allocation theories no longer apply. Without storage constraints one can *replicate* data to speed up access, and without bandwidth limitations, one can *flood* networks for fast search and downloading. These modern features both play a role in the *seamless* download scheduling paradigm of this paper. A general version of this problem consists of a given network  $(N, E)$  and a file  $F$  of  $M$  equal-size units, with  $F$  sited at some given node  $n_o$  (origin server) in the set  $N$  of network nodes. The links in  $E$  carry labels indicating (integer) link transmission delays. At time 0, an arbitrary but given subset of  $N - \{n_o\}$  acts as *clients* for  $F$  by sending requests along paths leading to  $n_o$ , requests being forwarded automatically by the nodes in the paths to  $n_o$ . The paradigm is online in the sense that nodes (including  $n_o$ ) along a path can start downloading cached units of  $F$  towards the requesting clients only after receiving requests to do so. In the CLIENT problem the subsets of  $F$  cached at the nodes must be such that (1) seamless download schedules occur at each client, i.e., the entire file arrives by time  $M$  at each client, and (2) the total cost is minimized, where cost is defined as the total amount of  $F$  cached at the nodes of the network. Note that the schedules of file units are in general out of order, with permutations varying from client to client. We describe a surprisingly simple solution to this problem when  $(N, E)$  is a tree rooted at  $n_o$ , with a scheduling protocol at each node that is independent of the problem instance. We then examine the more difficult ANY-CLIENT problem in which the clients are not known beforehand (not a part of the problem instance), but minimal-cost seamlessness is to be achieved no matter which nodes announce themselves as clients. We simplify the problem to path (chain) structures and give approximation algorithms which assume that each node knows its position in the path. Finally, we introduce an online, download flooding variant of ANY-CLIENT and again arrive at a simple optimal solution where the scheduling protocol at each node is independent of network structure. The number of open problems in seamless download scheduling is very large indeed because of the large number of interesting and practical assumptions that can be made in defining problem instances. We discuss a number of the more interesting cases.

# **Scheduling, a Critical Biography**

*Jan Karel Lenstra*

## **Tutorial Speakers**

**Shop Scheduling with Blocking**

*Peter Brucker*

**Stochastic Scheduling Subject to Preemptive-Repeat Machine  
Breakdowns**

*Xiaoqiang Cai*

**Hybrid Technologies for Supply Chain Planning and Scheduling**

*Alkis Vazacopoulos*

**Constraint Satisfaction**

*Edward Tsang*

**Supply Chain Scheduling**

*Nicholas Hall*

**Scheduling in Uncertain and Dynamic Environments**

*Stephen Smith*

# **Alternative Formulations**

Abstracts



# A TSP FORMULATION OF THE DNA SEQUENCING PROBLEM

Athanassios Nikolakopoulos and Haralambos Sarimveis

*Corresponding author: Haralambos Sarimveis, Assistant Professor at National Technical University of Athens, School of Chemical Engineering, e-mail: [hsarimv@central.ntua.gr](mailto:hsarimv@central.ntua.gr)*

*Nikolakopoulos Athanassios, PhD student at National Technical University of Athens, School of Chemical Engineering, e-mail: [thnikola@central.ntua.gr](mailto:thnikola@central.ntua.gr)*

*National Technical University of Athens, School of Chemical Engineering,  
9, Heroon Polytechniou Str. Zografou Campus, Athens 15780, Greece*

Key words: DNA sequencing with errors, Meta-heuristic Search, Traveling Salesman Problem, Threshold Accepting

## ABSTRACT

The main target of this paper is the formulation of the Sequencing by Hybridization problem (*SbH*) (Fodor et al., 1991) as an Asymmetric Traveling Salesman problem (*ATSP*) (Reinelt, 1994). Furthermore, a metaheuristic method is proposed for the solution of the problem. *SbH* is a method where an original DNA string of up to 1000 nucleotides is hybridized with a bio-chip that contains all sequences of nucleotides of length  $l$  ( $l \in \{8, 10, 12\}$ ). The nucleotides belong to the set: {adenine ( $a$ ), cytosine ( $c$ ), guanine ( $g$ ) and thymine ( $t$ )}. Thus, the whole DNA chain can be represented as a string comprised of a sequence of the four letters ( $a$ ,  $c$ ,  $g$  and  $t$ ) repeated in an arbitrary manner. The purpose of this bioinformatics method is to identify all the nucleotides and their related positions in the original string. All the identified sequences of length  $l$ , are called oligonucleotides and comprise the *spectrum*. Ideally, all the oligonucleotides are identical to fragments of the original sequence, overlap one over the immediately succeeding oligonucleotide by  $l-1 = 2$  nucleotides, so they reach a total number of oligonucleotides:  $n - l + 1$ . Unfortunately realistic spectrums contain errors. An error is called *negative* if an oligonucleotide from the ideal spectrum is not contained in the experimental spectrum, and *positive* if the reverse case occurs (Blazewicz et al., 1999). For example we assume a spectrum that comprises three oligonucleotides  $\{1, 2, 3\}$  of length  $l = 3$ , where  $1 := \{a, c, g\}$ ,  $2 := \{c, g, t\}$ ,  $3 := \{t, t, a\}$  and an original DNA string:  $\{c, g, t, a, c, g\}$ . Consider the arbitrary order of the oligonucleotides,  $order = [2 \ 3 \ 1]$ . The constructed DNA string is  $:= \{c, g, t, t, a, c, g\}$  consists of 7 nucleotides because the 2<sup>nd</sup> overlaps the 3<sup>rd</sup> and the 3<sup>rd</sup> overlaps the 1<sup>st</sup> oligonucleotide by one nucleotide. It is obvious that the spectrum is not ideal since not all

neighboring oligonucleotides overlap by  $l-1 = 2$  nucleotides. For example the 3<sup>rd</sup> oligonucleotide is overlapped by the 2<sup>nd</sup> by  $k = 1$  ( $t$ ) nucleotides and since  $l-1 = 2$ ,  $k < l-1$ . An example of a positive error is that the oligonucleotide ( $g t a$ ) is missing from the spectrum though it is contained in the original DNA sequence. An example of negative error is oligonucleotide 3:=( $t t a$ ) which is included in the spectrum though it is not contained in the DNA sequence. Such a data base of oligonucleotides that possibly contain errors must be processed appropriately to formulate a sequence of nucleotides that matches as close as possible the original DNA string. The bigger is the overlapping between successive oligonucleotides the bigger becomes the resemblance of the constructed DNA string to the original.

In the present work the oligonucleotides are considered as nodes of an ATSP. Since the nucleotide denoting letters of any two oligonucleotides  $i, j$  are known the maximum overlaps:

$$Ov_{ij} \quad i = 1, \dots, m \quad \text{and} \quad j = 1, \dots, m \quad (m: \text{total number of oligonucleotides})$$

of the last letters of  $i$  over the first letters of  $j$  can be easily calculated (see oligonucleotides 2 and 3 from the last example  $Ov_{12} = 2$  and  $Ov_{21} = 0$ ). Knowing the overlaps we compute the distances between the oligonucleotides by the following equation:

$$d_{ij} = 2 \cdot l - Ov_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, m$$

In general  $d_{ij} \neq d_{ji}$  (asymmetric problem  $d_{12} = 4 \neq d_{21} = 6$ ). The proposed solution is an order of oligonucleotides  $\mathbf{P} = \{P_1, P_2, \dots, P_m\}$ . The objective function of the problem is the minimization of the sum of the distances of sequential nodes in the proposed solution:

$$\text{Obj:} \quad \min \sum d_{P_i, P_{i+1}}, \quad i = 1, \dots, m$$

For the computational experiments we use DNA data from GenBank and produce erroneous spectrums for original strings containing 100 to 1000 nucleotides. The percentages of negative or/and positive errors in the produced spectrums vary from 10 to 30%. Before sequencing the oligonucleotides of the *spectrum*, the possible positive errors are recognized according to their maximum overlap and removed. If the procedure does not reach a feasible solution, the data corresponding to the less possible positive error are reinserted in the *spectrum* and the problem is resolved again. For the sequencing problem a variation of the Threshold Accepting (TA) (Dueck and Scheuer, 1990) method with intense Local Search (Salvesberg, 1992, Cirasella et al. 2001, Helsgaun, 2000, Lin and Kernighan, 1973) is used. The goal is to minimize the total traveled distance of the ATSP route, thus attaining the tightest bonding of the oligonucleotides and therefore the best identification to the original string. The current solution of the TA method is a permutation of the available oligonucleotides and the search is advanced by applying local moves (such as the  $k$ -Opt moves - Lin and Kernighan, 1973) in order to produce a new permutation. For each permutation (say [3 2 1] for example 1), an *Insertion method* constructs progressively a sequence of oligonucleotides ([1 2 3] with DNA string  $\{a, c, g, t, t, a\}$ ) by inserting them one by one in a position that corresponds to the minimal cost:

$$\arg\min_i (d_{ik} + d_{kj} - d_{ij}) \quad \text{when the oligonucleotide } k \text{ is inserted after } i$$

The solution [1 2 3] for example 1 is not correct but acceptable because the length of the constructed DNA string is equal to the length of the original string. The solution that corresponds to each permutation is evaluated by calculating the sum of all distances between successive oligonucleotides. TAs' iterative procedure is controlled by a problem-size *diminishing shell*. The *diminishing shell* is a

procedure that interrupts the execution of the TA algorithm at appropriate time instances. This procedure accomplishes a reduction of the size of the problem by grouping the tightly sequenced oligonucleotides into clusters of oligonucleotides, and formulates a new *ATSP* of smaller dimension. The nodes of this smaller *ATSP* are the oligonucleotide clusters. The entire procedure involving both the size diminishing shell and the TA algorithm is repeated till an acceptable solution (the length of the constructed DNA string is equal to the length of the original DNA string) is reached. The proposed method provides solutions of better quality compared to algorithms that have been presented in the recent bibliography (Blazewicz et al., 2002, Endo, 2004). Moreover it produces solutions for problems of bigger size i.e. DNA strings of bigger cardinality or/and with larger percentage of erroneous data.

## ACKNOWLEDGEMENTS

Financial support by FAGE S.A. and the General Secretariat of Research and Technology in Greece under the PENED 2001 research program (01EΔ38) is gratefully acknowledged.

## REFERENCES

1. Fodor, S.P.A., Read, J.L., Pirrung, M.C., Stryer, L., Lu, A.T., Solas, D. (1991). Light-directed, spatially addressable parallel chemical synthesis, *Science* 251 767–773.
2. Reinelt, G. (1994). The Traveling Salesman: Computational Solutions for TSP Applications, *Lecture Notes in Computer Science*, Springer-Verlag, Vol 810.
3. Blazewicz, J., Formanowicz, P., Kasprzak, M., Markiewicz, W.T. and Weglarz, J. (1999). DNA sequencing with positive and negative errors. *Journal of Computational Biology*, 6, 113–123.
4. Dueck, G., Scheuer, T. (1990). Threshold accepting. A general purpose optimization algorithm appearing superior to simulated annealing, *Journal of Computational Physics*, 90, 161–175.
5. Savelsbergh, (1992). The vehicle routing problem with time windows: minimizing route duration, *ORSA Journal on Computing*, 4, 146–154.
6. Cirasella, J., Johnson, D. S., McGeoch, L. A. and Zhang, W. (2001). The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In Proc. 3rd ALENEX, LNCS, Springer-Verlag, 2153, 32-59.
7. Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic, *European Journal of Operations Research*, 12, 106-130.
8. Lin, S., Kernighan, B.W. (1973). An Effective heuristic algorithm for the traveling salesman problem, *Operations Research*, 21, 972-989.
9. Blazewicz, J., Formanowicz, P., Guinand, F. and Kasprzak, M. (2002). A heuristic managing errors for DNA sequencing. *Journal of Bioinformatics*, 18(5), 652–660.
10. Endo, A. Takaho, (2004). Probabilistic Nucleotide Assembling Method for Sequencing by Hybridization, *Bioinformatics*, 20(14), 2181-2188.

# Multi-Criteria Soft Constraints in Timetabling

Hana Rudová, Miroslav Vlk

Faculty of Informatics, Masaryk University  
Botanická 68a, Brno 602 00, Czech Republic  
{hanka,xvlk@fi.muni.cz}

## 1 Introduction

Timetabling problem is an optimization problem where various criteria must be considered and where it is not possible to satisfy all of the requirements. We would like to show a framework which allows to include more criteria in a declarative style with the help of soft constraints. Each objective criteria can be represented by a set of soft constraints. Such set includes requirements of the same type, for example preferences on assignments of events to times or preferences on possible overlapping of events in time. This set can be associated with a weight representing importance of the criteria. The flexible assignment of the weight allows to emphasize or suppress particular criteria and obtain desirable solution.

Our work is motivated by a real timetabling problem at Purdue University, USA [7]. This large scale problem includes about 820 classes. About 90,000 course requests by almost 30,000 students must be considered. A major objective in developing an automated system is to minimize the number of potential student course conflicts. There are also various requirements which are not possible to satisfy and some optimization is needed. For example, back-to-back classes of one student or instructor should not be scheduled in two distant buildings, or we should allow for a balanced distribution of times for particular subjects.

Let us start with some basic definitions of constraint programming approach. A constraint satisfaction problem (CSP) [2] is a triple  $(V, D, C)$  where  $V$  is a finite set of variables,  $D$  is a set of possible values for variables (domain), and  $C$  is a finite set of constraints restricting the values of variables. A solution to a CSP is a complete assignment of the variables that satisfies all the constraints. It may not be possible to find a solution which would satisfy all of the (hard) constraints. In this case, we can consider soft constraints which may not be necessarily satisfied. Various approaches to soft constraints have been introduced and studied [1]. We will consider the weighted constraints [3] which are associated with some cost. The goal is to find such assignment of values to variables that the total cost of unsatisfied constraints is minimal. This approach includes optimization component but the multi-criteria decision making is not considered. Our goal is to show an extension of weighted soft constraints which is able to handle multiple criteria naturally.

We will first describe particular criteria and define their evaluation function. Next section will propose evaluation function for all criteria with the help of inconsistency counters. Section 4 specifies particular soft constraints and their propagation algorithms based on inconsistency counters. Some preliminary experiments are presented in Section 5.

## 2 Criteria

We can define unary soft constraints expressing importance of particular values in the domain of variable. In timetabling problem, two distinct criteria can be represented with the help of such soft constraints. The first criteria introduces a desirable *time placement* and the second criteria specifies a desirable *classroom placement* for classes. Both criteria are defined on each assignment  $\theta$  of values to variables and they sum up initial time preferences  $time(t_i, \theta(t_i))$  and initial room preferences  $room(r_i, \theta(r_i))$  for all classes  $i$ :

$$f_{time}(\theta) = \sum_{\forall i} time(t_i, \theta(t_i)) \ , \quad f_{room}(\theta) = \sum_{\forall i} room(r_i, \theta(r_i)) \ .$$

Other soft constraint can state that the two classes  $i$  and  $j$  represented by variables for start times share the  $c_{ij}$  students. Naturally the  $c_{ij}$  gives the cost of this constraint because  $c_{ij}$  students do not want scheduling of these classes at the same time. All soft constraints of this type are included in the *student conflict* criteria. To define it, we will expect that  $overlap(\theta(t_i), \theta(t_j)) = 1$  holds if  $i$  and  $j$  overlap in the assignment  $\theta$ . Otherwise  $overlap(\theta(t_i), \theta(t_j)) = 0$  holds. Now the student conflict criteria can be written as

$$f_{sc}(\theta) = \sum_{\forall i,j:i < j} c_{ij} \times overlap(\theta(t_i), \theta(t_j))$$

A similar constraint with additional variables for classrooms can express the *student distance* criteria. It means that the scheduling of the two classes  $i, j$  in consequent time periods (back-to-back classes) should discourage placement in very distant buildings because the  $c_{ij}$  students share these classes. We again need an additional function  $btb(\theta(t_i), \theta(t_j))$  which is equal either to 1 for back-to-back classes or to 0 for remaining classes. Function  $far(\theta(r_i), \theta(r_j))$  expresses how distant are two rooms  $i, j$ . For student conflicts, it corresponds to 0 if they are close enough and equals to 1 if they are too distant<sup>1</sup>. Now the student distance criteria is

$$f_{btb}(\theta) = \sum_{\forall i,j:i < j} c_{ij} \times btb(\theta(t_i), \theta(t_j)) \times far(\theta(r_i), \theta(r_j)) .$$

In our timetabling problem, classes of different subjects/departments like Math or Physics are included. A balanced distribution of times for particular departments is needed not to assign to one subject much worse or better times than to others (for example, one subject should not have all late afternoon lectures). The *department spread* criteria

$$f_{dept}(\theta) = \sum_{\forall D} \sum_{k=est}^{lst} \max(0, current(\theta, D, k) - limit(D)) .$$

is associated with the soft constraints which mean that at most  $limit(D)$  classes of one department  $D$  can be scheduled at the same time. Function  $current(\theta, D, k)$  computes the number of classes scheduled by  $\theta$  at each time unit of a day  $k$  from  $est$  to  $lst$ .

### 3 Inconsistency Counters

We have defined several criteria which needs to be extended to some soft constraints [1]. Now we would like to reformulate particular criteria to allow inclusion in constraint propagation algorithms for soft constraints [4]. Our current proposal is based on partial forward checking algorithm [3] which is the basic algorithm applying so called inconsistency counts. Inconsistency counters store information about the current number of violations for each value in the actual domain of the variable. Once all variables are assigned, only one value remains in the domain of each variable. Summarization of all remaining inconsistency counters (one per each variable) gives evaluation of the solution. In this section, we redefine all of the criteria using inconsistency counters. Consequently they can be used in constraint propagation algorithms of particular soft constraints.

Initially the value of inconsistency counter for time variable  $t_j$  and each its value  $a$  is set to  $time(t_j, a)$ . Similarly we initialize inconsistency counters for each room variable by  $room(r_j, b)$ .

Let  $\theta$  is some current partial assignment and  $A(\theta)$  is the set of variables assigned by  $\theta$ .  $V \ll v$  will denote that the variables from the set  $V$  were assigned earlier than the variable  $v$ .

Now the number of inconsistencies for the student conflict criteria can be written as

$$IC_{sc}(t_j, a) = \sum_{t_i \in A(\theta): \{t_i\} \ll t_j} c_{ij} \times overlap(\theta(t_i), a) .$$

The student distance criteria can propagate into inconsistency counter if only one of the variables  $t_i, r_i, t_j, r_j$  remains unassigned. Let us expect we have assigned  $t_i, r_i, r_j$ . Then we can derive

$$IC_{btb}(t_j, a) = \sum_{\{t_i, r_i, r_j\} \ll t_j} c_{ij} \times btb(\theta(t_i), a) \times far(\theta(r_i), \theta(r_j)) .$$

---

<sup>1</sup>Let us note that the function  $far$  could express various degrees of acceptability.

If one of the room variables ( $r_j$ ) is unassigned we have

$$IC_{btb}(r_j, a) = \sum_{\{t_i, t_j, r_i\} \ll r_j} c_{ij} \times btb(\theta(t_i), \theta(t_j)) \times far(\theta(r_i), a),$$

Now let us define inconsistency counters for the department spread criteria. Let  $periods(i, b)$  gives all time periods of the class  $i$  starting at time period  $b$ . The current number of classes  $j$  from the department  $D$  which were assigned before the class  $i$  using  $\theta$  to time period  $a$ , is

$$current(\theta, D, a, i) = \|\{j | j \in D \wedge \{t_j\} \ll t_i \wedge a \in periods(j, \theta(t_j))\}\|.$$

The limit value  $limit(D)$  for each department  $D$  can be compared with the maximal value of  $current(\theta, D, a, j)$  for all time periods of course  $j$  starting at  $a$

$$max\_current(j, a) = \max_{b \in periods(j, a)} \{current(\theta, D, b, j)\}$$

where  $j \in D$  holds. As a consequence we get

$$IC_{dept}(t_j, a) = \begin{cases} 0 & max\_current(j, a) < limit(D) \\ 1 & max\_current(j, a) = limit(D) \end{cases}$$

Unique inconsistency counter was defined for each variable and its value. Inconsistency counter for the time variable  $t_j$  and its value  $a$  equals to

$$IC(t_j, a) = w_{time} \times time(t_j, a) + w_{sc} \times IC_{sc}(t_j, a) + w_{btb} \times IC_{btb}(t_j, a) + w_{dept} \times IC_{dept}(t_j, a)$$

and, for the room variable  $r_j$  and its value  $b$ , it corresponds to

$$IC(r_j, b) = w_{room} \times room(r_j, b) + w_{btb} \times IC_{btb}(r_j, b).$$

$w_{time}, w_{sc}, w_{btb}, w_{dept}$ , and  $w_{room}$  are weights expressing importance of particular criteria.

Finally we obtain evaluation for each assignment  $\theta$

$$F(\theta) = \sum_{\forall j \text{ where } \theta(t_j)=a \wedge \theta(r_j)=b} IC(t_j, a) + IC(r_j, b)$$

## 4 Soft Constraints

We have implemented all of the criteria as the soft constraints of soft CLP( $FD$ ) solver [6]. This solver allows to maintain inconsistency counters for each time and room variable. Inconsistency counter is stored for each value which is in the current domain of the variable. It is initialized by the values of time placement and room placement criteria.

Student conflict criteria is implemented with the help of

`soft_disjunctives(T_i, D_i, ListT_j, ListD_j, ListC_ij)`

constraint posted for each class  $i$ . The class  $i$  is represented by the time variable  $T_i$  and its duration  $D_i$ . The lists `ListT_j` and `ListD_j` store time variables and durations for classes  $j$  which share some students with the class  $i$ . The list `ListC_ij` contains the list of numbers of shared students  $C_{ij}$  between class  $i$  and classes  $j$ . The solver for this constraint is activated once when the  $T_i$  is assigned a value. It increases inconsistency counters for time variables  $T_j$  from `ListT_j` which still have not assigned value and which could overlap with  $T_i$ . So, the values of  $T_j$  overlapping with  $T_i$  are computed and their inconsistency counters are increased by  $w_{sc} \times C_{ij}$ .

The soft constraint for student distance criteria has a similar structure to `soft_disjunctive` constraint

`soft_btb_dist(T_i, R_i, D_i, ListT_j, ListR_j, ListD_j, ListC_ij)`

Meaning of the variables remains to be same. In addition, we have corresponding room variables  $R_i$  and  $ListR_j$ . This soft constraint is activated two times. When the first of the variables  $T_i$  and  $R_i$  is instantiated we are looking for classes  $j$  which have assigned both  $T_j$  and  $R_j$ . When the second of the variables  $T_i$  and  $R_i$  is instantiated, the classes  $j$  with just one assigned variables from  $T_j$  and  $R_j$  are found. In all cases, we have three variables assigned and we can check for an increase one of the  $IC_{btt}$  counters.

Department spread criteria is implemented via

```
soft_dept(Dept, ListT_i, ListD_i, Limit)
```

constraint posted for each department  $Dept$ . The lists  $ListT_i$  and  $ListD_i$  contains time variables and durations of all classes from the department  $Dept$  and  $Limit$  is the limit value for the  $Dept$ . The constraint is activated any time some time variable  $T_i$  from  $ListT_i$  is instantiated. As a consequence, the current number of classes assigned during time periods of  $T_i$  is increased. Also inconsistency counters are increased for classes  $j$  from  $ListT_i$  which have assigned just  $Limit$  classes for some value. However, this increase could be processed for each value of each time variable just once.

## 5 Experiments

Let us present some of our preliminary experiments we achieved on Purdue Spring 2005 data set. This data set includes 821 classes and 50 classrooms. The total amount of joint enrollments is 97371.

The goal of our experiments was to set different weights to particular criteria and look for comparison among them. Both experiments at Figure 1 show evaluation for achieved time placement and student conflicts

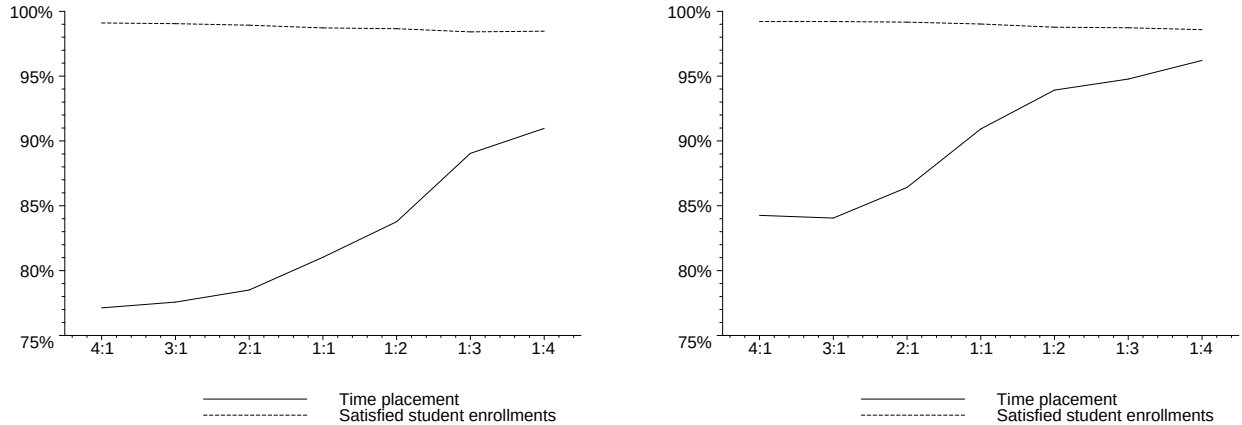


Figure 1: Comparison of criteria on time variables

(satisfied student enrollments corresponds to the supplement of student conflicts). Both criteria have set different weights  $w_{sc} : w_{time}$  ranging from 4 : 1 to 1 : 4. For example, 4 : 1 means that the student conflict criteria was four times more important than the time placement criteria. The first experiment at Figure 1 (left graph) shows the case where the weights for the student distance and department spread criteria are set to 1 (i.e.,  $w_{btt} = w_{dept} = 1$ ). The other experiment does not consider any of them (i.e.,  $w_{btt} = w_{dept} = 0$ ).

We can see that all criteria are strongly tight together. We are able to improve evaluation for time placement while worsening the number of student conflict and vice versa. Also we can see that the department spread criteria has a strong influence on the time placement. The department spread criteria is able to overweight some of the time placement preferences to achieve a better spread of classes of each department through the time.

## 6 Conclusion and Future Work

We have extended our soft  $CLP(FD)$  solver to handle various criteria from timetabling problems. This general proposal is based on inconsistency counters. It allows addition of other criteria by means of new types of soft constraints. We run some preliminary experiments on large scale Purdue timetabling problem. Our goal was to show that multi-criteria decision making could be implemented with the help of our soft constraints.

Our future work will consists in extension of constraint propagation algorithms for particular criteria. Next we will compare our results with local search-based solver for Purdue University timetabling problem [5].

## References

- [1] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, March 1997.
- [2] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
- [3] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
- [4] Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1–2):1–26, 2004.
- [5] Tomáš Müller, Hana Rudová, and Roman Barták. Minimal perturbation problem in course timetabling. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling, Selected Revised Papers*. Springer-Verlag LNCS, 2005. To appear.
- [6] Hana Rudová. Soft  $CLP(FD)$ . In Susan Haller and Ingrid Russell, editors, *Proceedings of the 16th International Florida Artificial Intelligence Symposium, FLAIRS-03*. AAAI Press, 2003.
- [7] Hana Rudová and Keith Murray. University course timetabling with soft constraints. In Edmund Burke and Patrick De Causmaecker, editors, *PATAT 2002 — Proceedings of the 4th international conference on the Practice And Theory of Automated Timetabling*, pages 73–89, 2002.



## MODELS FOR ADJACENT RESOURCE SCHEDULING

C.W. Duin and H.J. van der Sluis

*Operations Research Group*

*Faculty of Economics and Econometrics*

*University of Amsterdam*

Abstract:

We will formulate and compare models for an interesting relatively new problem, that of adjacent resource scheduling.

As an example, consider an airport where passengers check in for their flight, joining waiting lines before one or more desks; at the desk the luggage is checked and so forth. To smooth these operations the airport maintains a clear order in the waiting lines: a number  $n(j)$  of *adjacent* desks is to be assigned exclusively during a fixed time-interval  $I(j)$  to flight  $j$ . For each flight in a given time horizon, one seeks a feasible assignment to adjacent desks and the objective is to minimize the total number of involved desks.

We can formulate the problem more formally. A ‘bank’ of adjacent resource units  $r=1,2,\dots,R$  is to handle jobs  $j=1,2,\dots,J$  with starting period  $a(j)$  and completion period  $b(j)$  within a planning horizon with periods  $t=1,2,\dots,T$ . Each resource unit can

be assigned to at most one job per period. During an interval  $I(j)=[a(j),b(j)]$  (of  $b(j)-a(j)+1$  time periods in  $[1,T]$ ), each job  $j$  is to be assigned an invariant interval of  $n(j)$  resource units within  $[1,R]$ . The problem is to find the minimal number of resources  $R$  needed to complete all jobs.

On one hand, this problem can be compared to two-dimensional strip-packing where shifting is allowed in only one dimension, on the other hand it is like a reservation problem with additional adjacency constraints. In a more general problem variant the number  $n(j)$  varies over the time interval.

Firstly, we present two integer linear programs for these problems: a disjunctive model with  $r_{jt}$  representing the highest resource index for job  $j$  in period  $t$  as decision variable and an all-binary model with variables  $y_{jtr}$  equal to 1 if  $r$  is the highest resource index used by job  $j$  in period  $t$ . These models are trimmed to respectively a polynomial size and a pseudo-polynomial size. The latter is extended with a set of valid inequalities, and can then solve instances of moderate size.

Using the all-binary model, instances with 96 jobs and a planning horizon of 48 periods and constant  $n(j)$  are solved within a few minutes on a 400 MHz Pentium II, while the disjunctive model was able to solve only a few instances within reasonable time.

Secondly, we apply Constraint Programming (CP) and experiment with different variable selection and value assignment strategies. A straightforward disjunctive CP-model either solves instances with constant  $n(j)$  within seconds or not at all.

Finally, we compare the results of these different approaches for instances with  $n(j)$  varying over the time interval  $I(j)$ .

---

Key words: integer programming, constraint programming

# **Applications**

Full Papers

# Scheduling Multiple Types of Time-Shared Aircraft: Models and Methods for Practice

İtir Karaesmen <sup>\*</sup>      Pınar Keskinocak <sup>†</sup>      Sridhar Tayur <sup>‡</sup>      Wei Yang <sup>§</sup>

May 2005

## Abstract

Fractional aircraft ownership programs, where individuals or corporations own a fraction of an aircraft, have revolutionized the corporate aviation industry. Fractional management companies (FMCs) manage all aspects of aircraft operations enabling the owners to enjoy the benefits of private aviation without the associated responsibilities. We describe here the development of a decision support system for a leading FMC that considers multiple aircraft types as well as incorporates crew constraints. We developed several mathematical models and heuristics, and analyzed the models through a computational study using real data as well as randomly generated data. Our results indicate that the models are quite effective in finding optimal or near optimal solutions. The first phase of the implementation of one of these models at the FMC has led to a significant improvement in effective utilization of the aircraft (from 62% to over 70%), reduction of costs due to reduced empty moves, and hence increased profits. The second phase incorporating some of the other models is in progress, in large part due to the value created through the use of the phase 1 model.

---

<sup>\*</sup>İtir Karaesmen, Robert H. Smith School of Business, University of Maryland, College Park, MD, itir@umd.edu. Research supported by a grant from Division of Research and Graduate Studies at University of Maryland.

<sup>†</sup>Pınar Keskinocak, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, pinar@isye.gatech.edu. Pınar Keskinocak is supported by NSF Career Award DMI-0093844.

<sup>‡</sup>Sridhar Tayur, Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, stayur@andrew.cmu.edu

<sup>§</sup>Wei Yang, School of Business, Long Island University at C.W. Post, Long Island, NY, wei.yang@liu.edu.

# 1 Introduction

Fractional ownership programs have revolutionized the corporate aviation industry and created an exciting and rapidly growing market with a wide variety of customers, including celebrities, athletes, small and mid-sized companies who need business travels but cannot afford an entire aircraft. Some large corporations are also customers of these programs as they require supplemental flight hours in addition to their own corporate jets. According to The Wall Street Journal, the demand for fractional ownership programs has increased significantly over the past few years, and even more since the tragic events of September 11, 2001 [3]. By Spring 2002, the combined fleet of the three major fractional management companies (FMCs), NetJets, Flight Options, and Flexjet reached to 642 aircraft and the total shareholders to 5,200 from a few hundreds in late 1980s [4].

Travelers are often frustrated by commercial flights because of delays and disruptions, inflexible schedules, and safety concerns. For those looking for the convenience of a private jet and the freedom to walk away after every flight without being responsible for the aircraft, the crew, or any other aspect of aircraft operations, one option is to purchase a share of a specific aircraft from a fractional management company [6]. Under this concept, a  $1/8$  share entitles a customer to 100 hours of annual flight time and a  $1/2$  share licenses 400 hours [8] [16]. Requested trips are guaranteed within as short as 4 hours of notice before the departure time with access to about 5300 airports in the U.S. as opposed to only 563 served by scheduled carriers [10]. Owners have the flexibility to visit three different cities within one day and save hotel bills and overnight time away from home. This convenience and flexibility come at a price: There is a one-time ownership fee, a monthly management fee to cover maintenance, insurance, administrative and pilot expenses, and an hourly occupancy fee. For instance, in 2001 a  $1/8$  share of a Citation V Ultra which can seat up to 7 passengers was sold for \$0.75 million, plus \$8,196 per month and \$1,318 per hour whereas a  $1/8$  share of a Boeing Business Jet was sold for \$6 million plus \$41,480 per month and \$4,360 per hour [13].

Some regard FMCs as “unscheduled airlines” where the biggest challenge is orchestrating the operational requirements to meet the customers needs [15]. The aircraft and crew schedules, maintenance activities, and customer service all need to be coordinated for utmost customer satisfaction. Hence, for effective and efficient management of the rapidly growing fractional ownership programs, FMCs need to utilize computers and automated tools for their operational decisions - including aircraft scheduling, aircraft, crew and maintenance tracking, and customer support [15].

In this paper, we describe the models we developed to support the daily scheduling decisions of a major FMC. Other researchers have addressed similar applications in recent years [7][9][11]. Our contributions are four-fold. First, we extend the work of [7] by considering multiple types of aircraft as well as incorporating crew duty time constraints. Second, we complement the work of [9] and [11] by presenting alternative mathematical models and heuristics for the combined aircraft and crew scheduling problem. Third, we solve problems of significantly larger than the ones presented in [7] and [11] to address the current and anticipated future needs of the industry. Finally, the models we developed are tested on real world and randomly generated problem instances, which provide guidelines on when a model should be preferred and how effective each model is for an FMC. The first phase of the implementation at the FMC has already improved the operations significantly.

The paper follows the time line of our implementation experience at one of the largest FMCs, which does not have an internal Operations Research group. In fact, many of the practitioners at this FMC were initially skeptical of a decision support tool that relied on Operations Research, largely due their lack of experience with such tools. For this reason, we decided to break the project into two phases: a first phase, where a model and solution technique would be implemented rapidly at the production level for the schedulers to use. As

they became familiar with the tool, and the value created could be quantified, the skepticism was replaced with enthusiasm for more advanced features. At the same time, the FMC merged with another to double its size, thus requiring a superior formulation and solution technique. The success of phase 1 has led to phase 2 analysis and implementation.

The paper is organized as follows: Section 2 describes the challenges in aircraft and crew management. Section 3 introduces the decision support tool. We discuss the implementation and present the mathematical models for solving the aircraft scheduling problem in Section 4. Section 5 introduces the combined aircraft and crew scheduling problem; a heuristic and two exact models for the problem are discussed. The advantages and disadvantages of each approach are analyzed with computational experiments. Finally, we conclude in Section 6 and propose directions for future work.

## 2 Aircraft and Crew Management

To request a flight, a fractional owner calls the FMC and specifies the departure time, departure location and destination. The FMC checks the aircraft and crew availability and arranges transportation services. At any point in time, the aircraft in the fleet may be at several different locations or airborne, and the ones ready for service may have to be repositioned from their current locations to the trip departure locations to serve the customer. Although the customer pays for the actual travel time, the FMC bears the repositioning cost.

There are multiple types of aircraft in the fleet differing in speed, seating capacity and operational cost. Most customers own a particular type of aircraft together with other co-owners. When the owners of an aircraft type request services within the same time window, schedulers may have to “downgrade” or “upgrade” some customers to different types of aircraft based on their preferences. However, downgrades undermine owners’ goodwill while upgrades cost more for the FMC. Any trip request that cannot be met by company jets is satisfied by subcontracting to charter companies. Using a charter is significantly more expensive than using any of the company aircraft and the FMC bears the additional cost.

Not all the aircraft may be available to serve on a particular day because of periodic aircraft maintenance. Each aircraft must go through maintenance every three to five days. Maintenance takes place at specified locations and time periods determined by the FMC’s Maintenance Department. In addition to maintenance, aircraft schedules must be coordinated with crew duty time constraints under the Federal Aviation Administration (FAA) regulations. Therefore managing the aircraft and crew is a multi-dimensional game for an FMC with the objectives of (i) minimizing empty aircraft moves and subcontracts, (ii) satisfying maintenance restrictions, (iii) meeting FAA regulations for the crew, and (iv) meeting customer requests on time. Note that the scheduling problem faced by the FMC is “hard”: Even without any consideration of the crew, scheduling the aircraft with maintenance restrictions is NP-complete [7].

In many FMCs, due to flight safety concerns and the operational convenience of rotating the crew (see [9] for a detailed discussion), each aircraft and its crew are coupled together and regarded as a single entity for scheduling purposes. Hence, an aircraft is one operational unit that has to meet both maintenance restrictions and crew duty time limits. This approach is accepted by the FMC we worked with because of its consistency with the current practices in the industry and its simplicity in implementation.

Scheduling fractional ownership aircraft is first studied by [7]. The authors focus on a problem with only one type of aircraft that has maintenance and pre-scheduled trip restrictions. They show that the scheduling problem without pre-scheduled trips and maintenance restrictions is polynomially solvable, and the problem is NP-complete when either of these exists. They report computational results for problems with up to 30 aircraft and 100 trips and

compare the performance of a heuristic they developed to an exact mixed integer programming (MIP) solution computed by ILOG CPLEX. They do not consider crew related issues.

In [11] a decision support tool is developed for scheduling charter aircraft considering crew availability and multiple types of aircraft. The author proposes a set partitioning model to obtain optimal schedules where the cost of flying company aircraft, subcontracts and the penalties associated with idle company aircraft are to be minimized. To solve the problem, he generates a large set of good feasible candidate schedules for each aircraft, and uses a customized set partitioning solver. The effect of the length of the scheduling horizon is tested by solving problems for 24-hour, 36-hour and 48-hour long planning horizons, where 24-36 hours prove most effective. Computation times for at most 48 aircraft and 98 trips are reported.

The authors of [9] describe a decision support tool for fractional aircraft and crew scheduling where each aircraft and its crew are coupled. Schedules are restricted by crew daily work/rest periods and maintenance requirements. They model the problem as a 0-1 integer program, which, incorporated into a decision support tool, provides exact solutions using ILOG CPLEX. They discuss how the tool can be used to resolve problems and/or do sensitivity analysis by fixing aircraft-trip assignments. No computation times are reported.

### 3 Scheduling Decision Support Tool

Traditionally, the daily schedules for the aircraft were manually prepared by experienced schedulers in the company. To meet the needs for future expansion and become more competitive in the industry, the FMC decided to automate scheduling decisions and use an optimization based approach. Without an internal Operations Research group, the company does not have the ability to maintain sophisticated mathematical models and algorithms. A general purpose, commercial solver is preferred along with flexible, module based scheduling solutions that can be upgraded or enhanced easily when appropriate. The FMC focuses on simplicity in development, implementation and maintenance, and wants to stay away, to the extent possible, from customized, sophisticated heuristics.

A decision support tool – FlightOptimizer – has been developed for managing aircraft scheduling, maintenance tracking, and flight release. FlightOptimizer can create a daily schedule from scratch, manually modify an existing schedule to incorporate new trips and flight changes, and evaluate the cost of a schedule and the utilization rate of the fleet. The architecture of FlightOptimizer, which is designed to be highly flexible for future enhancements, is shown in Figure 1. FlightOptimizer consists of three modules – Optimizer Preparation Unit (OPU), Optimization Engine (OE) and Optimizer User Interface (OUI). Our team designed and developed OPU and OE. The OUI was developed by a boutique IT firm working closely with us and the schedulers. To create a feasible schedule, FlightOptimizer needs to be able to check the feasibility of including a requested trip or a group of trips into a schedule. OPU is used to accomplish this function. It retrieves the trip information, aircraft status and other related information from the central database server. It then processes the data and creates all the feasible trip assignments for an aircraft considering the positioning legs and required maintenance. Details of preprocessing are discussed in Section 3.1. Within OE there are two separate units: Model Creator (MC) and ILOG CPLEX solver. MC retrieves the information on feasible assignments from OPU and sets up the mathematical model for ILOG CPLEX. After a model is created, MC calls the routines from ILOG CPLEX, retrieves the output from the solver and passes the generated schedule to OUI. OUI presents the schedule via a user-friendly interface to the schedulers and dispatchers through client computers in the central operation control room. The interface shows the name of each aircraft in the fleet with its assigned trips during the scheduling horizon. In case of emergencies such as flight cancellation and aircraft breakdown, schedulers can manually change the schedules. They have the flexibility of trying

different scheduling scenarios before finalizing the most suitable one for the current situation.

A 24-hour planning horizon is adopted by the FMC due to the high uncertainty associated with flight requests outside this horizon. The practice in the company is to create a *Master Schedule* one day in advance since some airports need the flight and aircraft information at least 24 hours before the airport is used. Last minute travel requests or cancellations, as well as unexpected changes in the availability of aircraft due to mechanical problems all effect the existing Master Schedule. In this application, the OE is used to create the Master Schedule for a 24-hour period. The OUI, on the other hand, enables the schedulers to update the Master Schedule during the course of the horizon as new information becomes available.

### 3.1 Data Preparation Unit

The OPU is designed to preprocess the data to determine feasible aircraft-trip assignments. Each aircraft in the fleet is identified by its *tail number*. The earliest time a tail number is ready for service, its initial location and scheduled maintenance activities during the planning horizon are known a priori. For each trip we know (i) departure time, (ii) departure location, (iii) destination, (iv) type of aircraft owned by the customer, (v) any aircraft type or tail numbers that are incompatible with the requested trip, and (vi) whether or not the customer accepts chartered flights. Travel times between each pair of locations by each type of aircraft are also available.

Before formulating the scheduling problem, we first preprocess the input data to identify the aircraft-trip compatibility and the availability of a tail number to take a trip or a pair of trips back-to-back. We use index  $i$  to denote aircraft and  $j$  to represent trips. Assuming there are  $N$  aircraft in the fleet with  $K \geq 1$  different types and  $M$  requested trips, we store the preprocessed information in the matrices defined below.

$$\begin{aligned}
AQ(i, j) &= \begin{cases} 0, & \text{if tail number } i \text{ is incompatible with trip } j \text{ given customer preferences;} \\ -1, & \text{if } i \text{ is compatible but is a downgrade for } j; \\ 1, & \text{if } i \text{ is compatible and equal in type or an upgrade for trip } j. \end{cases} \\
CQ(j) &= \begin{cases} 1, & \text{if the customer requesting trip } j \text{ accepts a charter;} \\ 0, & \text{otherwise.} \end{cases} \\
AT(i, j) &= \begin{cases} 1, & \text{if } AQ(i, j) \neq 0 \text{ and tail number } i \text{ can serve the requested trip } j \text{ on time;} \\ 0, & \text{otherwise.} \end{cases} \\
ATT(i, j, k) &= \begin{cases} 1, & \text{if tail number } i \text{ can take trips } j \text{ and } k \text{ back-to-back;} \\ 0, & \text{otherwise.} \end{cases}
\end{aligned}$$

Matrix  $AQ$  of size  $N \times M$  records the information regarding the compatibility of a tail number with a requested trip. Vector  $CQ$  of size  $M$  contains the customer preferences with respect to chartered flights. Matrix  $AT$  of size  $N \times M$  records the information regarding whether tail number  $i$  can serve trip  $j$ . Matrix  $ATT$  of size  $N \times M \times M$  summarizes the information on whether a tail number can serve two or more trips back-to-back. These matrices are passed to the OE to setup the mathematical model. Preprocessing dramatically reduces the number of decision variables by eliminating unacceptable/infeasible assignments. It can be completed very easily and the computation time is polynomial in  $N$  and  $M$ .

As mentioned before, each aircraft has to undergo pre-scheduled maintenance. One approach for modelling maintenance is to represent each maintenance actually as a “special trip” as in [7] by defining the duration of the maintenance as travel time, and the maintenance location as the origin and the destination of the trip. This special trip should be assigned to the specified tail number. An alternative approach is to incorporate the maintenance information



in the preprocessing. Since each aircraft undergoes at most one scheduled maintenance in a 24-hour period, and only a fraction of the fleet needs maintenance on the same day, we have chosen the latter approach. The matrices  $AT$  and  $ATT$  are computed accordingly to ensure that an aircraft-trip assignment is deemed feasible only if the aircraft can complete the trip in the desired time frame without violating its scheduled maintenance.

We aim to minimize the total cost in a 24-hour period while ensuring that each trip is served by either a tail number or a charter. Total cost consists of (i) repositioning costs, (ii) subcontracting costs and (iii) “hidden costs” that are associated with the use of an aircraft. The hidden costs represent the cost of flying the aircraft while serving a trip and the upgrade/downgrade costs associated with that aircraft-trip assignment. We use the following notation.

- $c_{i0j}$  = the cost of repositioning tail number  $i$  from its initial location to the departure location of trip  $j$ ,  $i = 1, \dots, N$ ,  $j = 1, \dots, M$ .
- $c_{ijk}$  = the cost of repositioning tail number  $i$  from the destination of trip  $j$  to the departure location of trip  $k$ ,  $i = 1, \dots, N$ ,  $j, k = 1, \dots, M$ .
- $b_j$  = the cost of subcontracting trip  $j$ ,  $j = 1, \dots, M$ .
- $a_{ij}$  = the “hidden” cost while the customer occupies the aircraft,  $i = 1, \dots, N$ ,  $j = 1, \dots, M$ .
- $l$  = the load factor,  $l = M/N$ .

For a trip (customer) that does not accept charters, i.e.,  $CQ(j) = 0$ , we set  $b_j = B_\infty$  where  $B_\infty$  denotes a high penalty cost. If  $AQ(i, j) = 1$ , then  $a_{ij}$  equals to only the operational cost, i.e.,  $a_{ij}$  is the cost of flying tail number  $i$  while serving trip  $j$ . If  $AQ(i, j) = -1$ , then tail number  $i$  is a downgrade for trip  $j$ . Hence,  $a_{ij}$  includes additional downgrade penalty and should be higher than the operational cost; however, it is not as high as  $b_j$ . Therefore, downgrades are penalized, but are preferred over charters.

### 3.2 Models for the Optimization Engine

We developed four mathematical models. The first two models, called NETIP and MIPS, solve the aircraft scheduling problem without consideration of the crew. NETIP is a minimum cost network flow model with side constraints. It is easy to build and incorporate into ILOG CPLEX because of the special structure of the constraint matrix, and can solve problems with small fleet size and low load factor efficiently. However, the advantage in terms of computation time diminishes as the fleet size increases. Therefore, as an alternative, we evaluate MIPS, a mixed integer programming model. Both models are presented in Section 4.

We evaluate a heuristic and two exact mathematical models for the combined aircraft and crew scheduling problem in Section 5. The heuristic - called RESTORE - enforces crew restrictions on the aircraft by adjusting a solution obtained by NETIP or MIPS. The first mathematical model is a mixed integer programming model that creates feasible and optimal solutions for both aircraft and crew by incorporating additional constraints to MIPS. We call this model MIPS-CREW. Both RESTORE and MIPS-CREW assume that the crew need only one rest during the planning horizon (which is the case for a 24-hour planning horizon). In addition, the computation time of MIPS-CREW is usually higher than that of MIPS. With the goal of overcoming the limitations of MIPS-CREW and RESTORE, we propose a set partitioning model called BP, which is solved using branch-and-price. These models and methods provide optimal or near-optimal solutions for the combined aircraft and crew scheduling problem.

We implemented NETIP for the FMC in the first phase because of its advantages in the development of the OE, and its computational performance in practice. It has been used by the company successfully since Spring 2002. A key performance indicator, measured carefully

both by schedulers as well as senior management is *utilization*, defined as

$$\text{Utilization} = \frac{\text{Total flight time with passengers}}{\text{Total flight time with and without passengers}}.$$

Since the implementation, the utilization has increased to over 70% from 62%. Since FlightOptimizer is designed to be a module-based decision support tool, alternatives models such as MIPS, MIPS-CREW and BP can be easily incorporated into the OE to accommodate future operational requirements.

## 4 Aircraft Scheduling

We present two mathematical models for the aircraft scheduling problem. Both are MIP formulations that determine the best aircraft-trip assignments in a given planning horizon.

### 4.1 Minimum Cost Network Flow Model

In this section, we present NETIP, a minimum cost network flow model with side constraints for solving the aircraft scheduling problem. We construct a directed acyclic graph  $G = (V, A)$  where  $V$  denotes the set of vertices and  $A$  represents the set of arcs (Figure 2(a)). The information from the OPU is used to construct the graph. The nodes and arcs of  $G$  are defined as follows:

**Nodes:**

A dummy source node  $s$  and a dummy sink node  $t$ .

A node  $p_i$  for tail number  $i$ ,  $i = 1, \dots, N$ .

Two nodes  $u_{ij}$  and  $v_{ij}$  for tail number  $i$  and trip  $j$ ,  $i = 1, \dots, N$  and  $j = 1, \dots, M$ .

A node  $r_j$  for trip  $j$ ,  $j = 1, \dots, M$ .

**Arcs:**

$(s, p_i)$  and  $(s, u_{ij})$  at cost 0,  $i = 1, \dots, N$  and  $j = 1, \dots, M$ .

$(r_j, t)$  at cost 0,  $j = 1, \dots, M$ .

$(p_i, v_{ij})$  for  $AT(i, j) = 1$  at cost  $c_{i0j}$ ,  $i = 1, \dots, N$  and  $j = 1, \dots, M$ .

$(u_{ij}, v_{ik})$  for  $ATT(i, j, k) = 1$  at cost  $c_{ijk}$ ,  $i = 1, \dots, N$ ,  $j = 1, \dots, M$  and  $k = 1, \dots, M$ .

$(s, r_j)$  for trips  $j$  at cost  $b_j$ ,  $j = 1, \dots, M$ .

$(v_{ij}, r_j)$  for trips  $j$  at cost  $a_{ij}$ ,  $i = 1, \dots, N$  and  $j = 1, \dots, M$ .

Each arc has a capacity of one. On this graph, we can easily determine the minimum cost flow of value  $M$  from node  $s$  to node  $t$ . A positive flow on arc  $(p_i, v_{ij})$  indicates that the first trip taken by tail number  $i$  is trip  $j$ . A positive flow on arc  $(u_{ij}, v_{ik})$  means that trip  $j$  is taken immediately before  $k$  by tail number  $i$ . A positive flow on arc  $(s, r_j)$  indicates that trip  $j$  is subcontracted. Arcs to/from node  $r_j$  ensure that trip  $j$  is taken by only one tail number or is subcontracted. Solving the minimum cost network flow problem on the given graph provides an integer solution. Unfortunately, this solution does not correspond to a solution to the aircraft scheduling problem unless side constraints are added. Consider the example shown in Figure 2 with 2 aircraft and 3 trips. Assume that the optimal solution to the minimum cost network flow problem in Figure 2(a), is given by routes  $s \rightarrow p_1 \rightarrow v_{11} \rightarrow r_1 \rightarrow t$ ,  $s \rightarrow u_{11} \rightarrow v_{12} \rightarrow r_2 \rightarrow t$  and  $s \rightarrow u_{22} \rightarrow v_{23} \rightarrow r_3 \rightarrow t$ . Hence, all the trips are assigned to the aircraft. Interpreting the solution in Figure 2(b), one sees that tail number 1 takes trip 1 as its first trip, followed by trip 2. On the other hand, trip 3 is assigned to tail number 2. Although this is a feasible schedule, it need not be optimal since the cost of the network flow solution is not the actual cost of the schedule. In particular, the cost computation assumes that trip 2 precedes trip 3 (on tail number 2) but that is not the case. Therefore, such solutions need to be eliminated by adding side constraints.

Side constraints are added to ensure that for each aircraft that serves trips  $j$  and  $k$  back-to-back, trip  $j$  is either the first trip of that aircraft or it is preceded by another trip  $q$ . Unfortunately, these side constraints destroy the total unimodularity of the constraint matrix, i.e., the linear programming (LP) relaxation of the model is not guaranteed to have an integer solution. Therefore, one has to work with a 0-1 integer programming formulation. Let  $x(e, f)$  to be the flow from node  $e$  to node  $f$  on arc  $(e, f)$  of graph  $G$ . We provide the NETIP formulation below.

$$\text{Min} \quad \sum_{i=1}^N \sum_{\{j: AT(i,j)=1\}} c_{i0j} \cdot x(p_i, v_{ij}) + \sum_{i=1}^N \sum_{\{j: AT(i,j)=1\}} \sum_{\{k: ATT(i,j,k)=1\}} c_{ijk} \cdot x(u_{ij}, v_{ik}) \quad (1)$$

$$+ \sum_{i=1}^N \sum_{j=1}^M a_{ij} \cdot x(v_{ij}, r_j) + \sum_{j=1}^M b_j \cdot x(s, r_j) \quad (2)$$

*s.t.*

$$\sum_{i=1}^N x(s, p_i) + \sum_{j=1}^M x(s, r_j) + \sum_{i=1}^N \sum_{j=1}^M x(s, u_{ij}) = M \quad (3)$$

$$\sum_{j=1}^M x(r_j, t) = M \quad (4)$$

$$x(s, p_i) - \sum_{\{j: AT(i,j)=1\}} x(p_i, v_{ij}) = 0, \quad i = 1, \dots, N \quad (5)$$

$$x(s, u_{ij}) - \sum_{\{k: ATT(i,j,k)=1\}} x(u_{ij}, v_{ik}) = 0, \quad \forall i, j \text{ with } AT(i, j) = 1 \quad (6)$$

$$x(s, r_j) + \sum_{i=1}^N x(v_{ij}, r_j) - x(r_j, t) = 0, \quad j = 1, \dots, M \quad (7)$$

$$x(p_i, v_{ik}) + \sum_{\{j: ATT(i,j,k)=1\}} x(u_{ij}, v_{ik}) - x(v_{ik}, r_k) = 0, \quad k = 1, \dots, M, \quad \forall i \text{ with } AT(i, k) = 1 \quad (8)$$

$$x(s, u_{ij}) - x(v_{ij}, r_j) \leq 0, \quad i = 1, \dots, N, \quad j = 1, \dots, M \quad (9)$$

$$x(e, f) \in \{0, 1\}, \quad \forall (e, f) \in A. \quad (10)$$

The objective is to minimize total costs. The two terms in (1) are the repositioning costs, whereas the first term in (2) represents the hidden costs of operation and the second one represents the subcontracting cost. Constraints (3) - (8) ensure the flow balance of each node in the network, and (9) are the side constraints. The side constraints ensure that there is no flow from  $s$  to  $u_{ij}$  unless there is positive flow from  $v_{ij}$  to  $r_j$ , i.e., trip  $j$  cannot precede another trip on tail number  $i$  unless trip  $j$  is assigned to  $i$ .

Let  $\|AT\|$  and  $\|ATT\|$  denote the number of non-zero coefficients in the  $AT$  and  $ATT$  matrixes respectively. NETIP has  $(N + 2M + 3\|AT\| + \|ATT\|)$  variables and  $(2 + N + 3M + \|AT\|)$  constraints. The number of side constraints in (9) is  $NM$ , which increases with the fleet size and the number of requested trips. The main advantages of NETIP are the ease of its implementation using ILOG CPLEX and its underlying network structure. However, for large fleet sizes and/or high load factors, side constraints dominate the network constraint matrix and affect the solution time negatively. We present an alternative formulation to deal with this issue.

## 4.2 Mixed Integer Programming Model

In this formulation (we call it MIPS) we use the following decision variables.

$$\begin{aligned}
S_j &= \begin{cases} 1, & \text{if trip } j \text{ is subcontracted} \\ 0, & \text{otherwise} \end{cases}, \quad j = 1, \dots, M. \\
Z_{ijk} &= \begin{cases} 1, & \text{if tail number } i \text{ serves trip } j \text{ just before trip } k \\ 0, & \text{otherwise} \end{cases}, \quad \forall i, j, k \text{ if } ATT(i, j, k) = 1. \\
Z_{i00} &= \begin{cases} 1, & \text{if tail number } i \text{ does not serve any trips} \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, \dots, N. \\
Z_{i0k} &= \begin{cases} 1, & \text{if trip } k \text{ is the first trip served by tail number } i \\ 0, & \text{otherwise} \end{cases}, \quad \forall i, k \text{ if } AT(i, k) = 1. \\
Z_{ij0} &= \begin{cases} 1, & \text{if trip } j \text{ is the last trip served by tail number } i \\ 0, & \text{otherwise} \end{cases}, \quad \forall i, j \text{ if } AT(i, j) = 1.
\end{aligned}$$

We provide the MIPS formulation below.

$$\begin{aligned}
(MIPS) \quad \text{Min} \quad & \sum_{i=1}^N \sum_{j=1}^M c_{i0j} \cdot Z_{i0j} + \sum_{i=1}^N \sum_{j=1}^M \sum_{k=1}^M c_{ijk} \cdot Z_{ijk} \\
& + \sum_{i=1}^N \sum_{j=1}^M \sum_{k=0}^M a_{ij} \cdot Z_{ijk} + \sum_{j=1}^M b_j \cdot S_j
\end{aligned} \tag{11}$$

$$s.t. \quad S_k + \sum_{i=1}^N \sum_{j=0}^M Z_{ijk} = 1, \quad k = 1, \dots, M. \tag{12}$$

$$\sum_{k=0}^M Z_{ijk} - \sum_{q=0}^M Z_{iqj} = 0, \quad \forall i, j \text{ when } AT(i, j) = 1. \tag{13}$$

$$\begin{aligned}
& \sum_{k=0}^M Z_{i0k} = 1, \quad i = 1, \dots, N. \\
& Z_{ijk} \in \{0, 1\}, \quad S_j \in \{0, 1\}.
\end{aligned} \tag{14}$$

Similar to NETIP, the objective (equation (11)) is to minimize the total cost of repositioning legs, hidden cost of operations, and subcontracting costs. Constraints (12) ensure that any customer trip has to be served by a charter or by a company aircraft. Constraints (13) ensure that if trip  $k$  is served right after trip  $j$ , then trip  $j$  is either the first trip of that tail number or it is preceded by another trip  $q$ . Constraints (14) ensure that each tail number has a first trip, or no trips at all. MIPS is related to the models presented in both [7] and [9]. MIPS considers multiple types of aircraft and upgrade/downgrade, and hence generalizes the single type aircraft model of [7]. However, as opposed to [7], MIPS does not consider maintenance restrictions. On the other hand, the model in [9] considers scheduling multiple types of aircraft subject to crew availability and MIPS is a special case of that model.

MIPS has  $(N + M + 2\|AT\| + \|ATT\|)$  decision variables and  $(N + M + \|AT\|)$  constraints, significantly fewer than NETIP. Hence, it can potentially perform better especially in problems with large fleet sizes and/or load factors.

### 4.3 Comparison of NETIP and MIPS

We have tested NETIP and MIPS using both real world and randomly generated data sets. The results in Table 1 show the performance of the models on real data which was obtained during the winter season of 2001. The load factors range from 0.66 to 2.08 for the given instances. As we introduced in Sections 4.1 and 4.2 the number of decision variables and constraints are functions of  $N$ ,  $M$ ,  $\|AT\|$  and  $\|ATT\|$ . For instance, when  $(N, M) = (68, 135)$  there are 36,609 decision variables and 4,770 constraints in NETIP, and 32,179 decision variables and 4,498 constraints in MIPS. The column “Setup Time” denotes the time spent to retrieve the preprocessed information and to prepare the formulations for CPLEX. The “Solution Time” denotes the time for CPLEX solver to determine an optimal integer solution. The problems are solved using ILOG CPLEX 7.5 on a PC with 1.13 GHz CPU and 512 MB memory, and all computation times are reported in CPU seconds. The setup time of NETIP is shorter than MIPS while NETIP requires longer solution time. The total computation time of NETIP (the sum of setup time and solution time) during the trial period was no more than 12 seconds, and in most cases less than 2 seconds, which is comparable to MIPS.

Although the results in Table 1 show that the run time of OE is satisfactory when either NETIP or MIPS is used, this is not necessarily representative of the performance for future use. With the expected growth of the fleet, the daily scheduling problems at the FMC will be of significantly larger size. In addition, load factors can be as high as 2 to 3 during high seasons. To examine the effect of fleet size and load factors on the computation time, we tested NETIP and MIPS using randomly generated data sets.

The data is generated as follows: We first chose 100 cities that are uniformly distributed on a 100 by 100 grid. The flight time between two adjacent points on the grid is assumed to be 3 minutes, therefore the longest flight is about 7 hours. Using the 100 cities, we randomly select (i) the initial location of the aircraft and (ii) a departure location and destination location for each trip with flight time greater than 30 minutes. The departure time for a trip is uniformly distributed between 0 and 900 minutes. The earliest available time of each tail number is uniformly selected between 0 and 600 minutes. There are 8 types of aircraft in the fleet and an aircraft type is randomly selected for each trip. During the planning horizon a randomly selected 20% of the fleet must go through maintenance services. Each maintenance service lasts 150 minutes. The data sets generated by this design mimic the FMC’s operations in a 24-hour period.

We use five different fleet sizes  $N = 10, 30, 50, 70, 100$  in our experiments. Each fleet size is tested using load factors of 1, 2 and 3, yielding a total of 15 experiments. We generate 10 independent, random data sets for each experiment. We report the average performance with respect to each experiment in Table 2.

The last two columns in Table 2 show the average number of nodes accessed by the branch-and-bound (B&B) scheme used in ILOG CPLEX. Because of its special structure, NETIP needs smaller number of nodes in a B&B tree in most cases. In 91% of the random data sets tested, the LP relaxation of at least one of the formulations solves the problem optimally (i.e., zero B&B nodes). Among those data sets, there are a few cases where the LP relaxation of MIPS yields the integer solution while NETIP fails to do so, and vice versa. We infer from our experiments that neither formulation is tighter than the other.

The solution time varies dramatically with fleet size and load factor. With small fleet sizes and low load factors, NETIP is slightly faster than MIPS. When fleet size and load factor increase, MIPS has a significantly faster solution time, and a better total computation time overall even though its setup time is higher than NETIP. For example, when  $N = 100$  and  $M = 300$ , it takes around 16 minutes for MIPS to generate the optimal solution compared to 36 minutes by NETIP. Therefore, MIPS could be a better choice than NETIP for the FMC in the future since the fleet size is expected to increase further.

## 5 Combined Aircraft and Crew Scheduling

Under FAA regulations the duty period of a crew cannot exceed more than 14 hours (we call it *DUTY*). Each duty period must be followed by a rest period of at least 10 consecutive hours. The start time of a duty period is defined as when a crew serves its first flight of the day (which can be a customer trip, or a repositioning flight), rather than the time a crew ends its previous rest period and becomes ready for service. Therefore, while creating a schedule, only those trips that can be served within *DUTY* hours since the first flight departure of an aircraft are eligible to be assigned to that aircraft. This restriction can be easily handled at the preprocessing stage if the duty start time of each crew is known. For instance, if a tail number starts its duty at 9 a.m. and  $DUTY = 14$  hours, the information in matrix  $AT$  can be updated easily to make sure that the tail number does not take any trips that require flying past 11 p.m. Unfortunately, the only information that is available prior to creating a schedule is when the crew first becomes available for service. The actual duty start time of a crew-aircraft pair depends on the first trip assigned to that aircraft.

In this section, we present a heuristic and two exact models to solve the scheduling problem when aircraft are subject to crew availability. First we present a heuristic based on the idea that MIPS (or NETIP) solutions can be “corrected” in a simple manner. The first mathematical model enhances MIPS by adding explicit constraints on crew duty time. The second model is a set partitioning formulation. These enhanced approaches fit the architecture of the FlightOptimizer and can easily be incorporated into the OE.

### 5.1 A Heuristic Based on Simple Correction to MIPS

The heuristic is based on the idea that the solution to MIPS (or NETIP) will more likely be feasible for the combined aircraft and crew scheduling problem, if the crew availability can be appropriately preprocessed in the OPU. Suppose there is a service period of  $\overline{DUTY}$  hours for each aircraft. This spans the period when the crew is “available”, i.e., it includes the duty time as well as the idle time before duty, when the crew is available but not used for service. Matrix  $AT$  can be updated accordingly: Starting from the earliest time a tail number is available, it will remain in service for the next  $\overline{DUTY}$  hours, and cannot take any trips that require flying beyond this duty period. The main advantage of this heuristic is that it is easily incorporated into NETIP and MIPS without affecting the computation times to create a Master Schedule. The disadvantage is that the schedule obtained may violate the actual crew duty time limits or impose tighter than necessary duty constraints on the aircraft. Choosing a conservative value for  $\overline{DUTY}$ , say  $\overline{DUTY} = 14$  hours, will ensure schedule feasibility, but possibly lead to lower aircraft utilization and higher cost. On the other hand, longer  $\overline{DUTY}$  results in a more cost-efficient schedule, but at a higher risk of infeasibility. We tested different  $\overline{DUTY}$  from 14 to 16 hours with increments of half an hour. Our computational results indicate that  $\overline{DUTY} = 14.5$  hours achieve the best tradeoff between schedule feasibility and schedule efficiency in terms of total operational cost.

The heuristic - called RESTORE - restores the feasibility of a schedule created by MIPS (or NETIP) that uses preprocessed data based on a preselected  $\overline{DUTY}$  value. Here is the description:

- Step 1:* Using  $\overline{DUTY}$ , preprocess the data, solve the problem and get Master Schedule.
- Step 2:* If the Master Schedule obtained in Step 1 is feasible, goto step 5.
- Step 3:* Undo all the aircraft-trip assignments that violate  $DUTY$ .  
Append all of these released trips to the *restore queue*.
- Step 4:* For each trip in the *restore queue*,  
Assign the trip to the aircraft which can take the trip without exceeding  $DUTY$  at the lowest cost while retaining its existing assignments.  
If no such aircraft exists, subcontract the trip.
- Step 5:* Master Schedule is feasible, pass the related information to OUI.

Notice that the computation time of RESTORE is simply determined by MIPS (or NETIP) since all the steps except Step 2 have polynomial running time (and completed within seconds in our experiments). The heuristic is not only useful to create an aircraft and crew feasible Master Schedule, but also practical for schedulers: They can update the Master Schedule during the course of the day manually following Steps 4 and 5 of RESTORE based on new information. The OUI of FlightOptimizer enables schedulers to make necessary adjustments in an effective way.

## 5.2 A MIP Model With Hard Constraints on Crew Duty Times

In a 24-hour planning horizon, there is only one duty period for the crew followed by a rest period. Let  $stime(j)$  be the departure time of trip  $j$ ,  $ttime(i, k)$  be the total travel time for aircraft  $i$  taking trip  $k$  and  $atrepo(i, j)$  be the repositioning time from the location of aircraft  $i$  to the departure of trip  $j$ . The constraints

$$\sum_{\{k: AT(i, k)=1\}} [stime(k) + ttime(i, k)] \cdot Z_{ik0} - \sum_{\{j: AT(i, j)=1\}} [stime(j) - atrepo(i, j)] \cdot Z_{i0j} \leq DUTY \quad (15)$$

are added to MIPS for each  $i = 1, \dots, N$  to ensure that the time between the first departure of the day (i.e., the time aircraft starts service) and the last landing of the day (i.e., the time aircraft completes service) does not exceed  $DUTY$  hours. We call this new MIP formulation MIPS-CREW.

## 5.3 A Set Partitioning Based Formulation

An alternative approach for solving the aircraft scheduling problem with crew duty time restrictions is to use a set partitioning formulation. The idea is to assign tail numbers to feasible routes to ensure that each tail number is assigned to at most one route and each trip is assigned to a tail number or a charter. The set partitioning approach is very appealing for problems where costs are nonlinear and discrete, and complicated operational rules are imposed. One such problem is crew scheduling by commercial airlines [1], which is similar to the combined aircraft and crew scheduling problem faced by FMCs. In both cases, “resources” (aircraft in case of FMCs and crew in case of commercial airlines) have to be assigned to a set of trips, and all trips have to be covered. However, there are three major differences: (i) Each crew schedule in commercial airlines starts and ends at the same crew base, which is the city where the crew is stationed. In FMCs, even though the crew travel to their home bases periodically, they can come on or off duty at any location because of the nature of the business. The travel from a home base to the location where the duty is to start is usually provided by means other than the company aircraft. (ii) Commercial airlines aim to minimize the costs related to long or frequent layovers within a duty period, long overnight rests away from base, and reposi-

tioning. FMCs minimize the repositioning cost, subcontracting cost and hidden costs. The repositioning is rare in commercial airlines while it is common in FMCs. (iii) Subcontracting is not an option in commercial airlines.

We use the following notation in our formulation:

$$\begin{aligned}
\Omega^i &= \text{the set of feasible routes for tail number } i, \quad i = 1, \dots, M + N. \\
p &= \text{a feasible route, } p \in \Omega^i. \\
c_{ip} &= \text{the total cost of taking route } p \text{ by tail number } i, \quad i = 1, \dots, M + N \text{ and } p \in \Omega^i. \\
a_{jp}^i &= \begin{cases} 1, & \text{if trip } j \text{ is on route } p \\ 0, & \text{otherwise} \end{cases}, \quad p \in \Omega^i, \quad i = 1, \dots, M + N \text{ and } j = 1, \dots, M. \\
\theta_{ip} &= \begin{cases} 1, & \text{if tail number } i \text{ takes route } p \\ 0, & \text{otherwise} \end{cases}, \quad p \in \Omega^i \text{ and } i = 1, \dots, M + N.
\end{aligned}$$

Note the tail numbers are indexed from 1 to  $N + M$  where the last  $M$  tail numbers denote charters as opposed to company aircraft. Each tail number  $N + i$  has only one feasible route which is associated with trip  $i$ , and the cost of the route is the charter cost of that trip. Feasible routes for the company aircraft can be determined using the same preprocessing rules used by MIPS (see Section 3.1) ensuring that the crew duty limits are not exceeded when a number of trips are assigned to an aircraft.

The Master Problem (MP) is

$$\begin{aligned}
(MP) \quad & \text{Min} \quad \sum_{i=1}^{M+N} \sum_{p \in \Omega^i} c_{ip} \theta_{ip} \\
& \sum_{i=1}^{M+N} \sum_{p \in \Omega^i} a_{jp}^i \theta_{ip} = 1, \quad j = 1, \dots, M \tag{16}
\end{aligned}$$

$$\sum_{p \in \Omega^i} \theta_{ip} \leq 1, \quad i = 1, \dots, M + N \tag{17}$$

$$\theta_{ip} \in \{0, 1\}, \quad i = 1, \dots, M + N, \text{ and } p \in \Omega^i. \tag{18}$$

where constraints (16) ensure that each trip is taken either by one tail number or by a charter and constraints (17) ensure that a tail number is assigned to at most one route.

One possible approach (we call it SP) for solving MP is to generate all feasible routes and then solve the model as a standard integer program. There are two drawbacks of this approach. The number of feasible routes might be quite high, hence, generating all of them may not be computationally feasible. Even if the routes could be generated, the resulting model might not be solvable in a reasonable time with a commercial solver, given the high number of 0-1 variables. Given these drawbacks of SP, the author of [11] follows a slightly different approach for solving the set partitioning model for charter aircraft scheduling problem. He generates only a subset of the feasible routes, and then uses a customized solver. While it is computationally more efficient than SP, this approach does not guarantee optimality. Another approach for solving the (large scale) set partitioning problems is branch-and-price (BP), which combines column generation and branch-and-bound (B&B). Next, we discuss the BP approach we used for solving the combined aircraft and crew scheduling problem.

### 5.3.1 Branch and Price Method

The standard approach to solving integer programming problems is branch-and-bound (B&B). At each branch of the B&B tree, we solve the LP relaxation of the integer problem. However, given the large number of potential columns (variables) in the MP, generating all columns upfront and solving the relaxation of MP exactly might be challenging. A computationally



more efficient approach is column generation. The general idea of column generation is to start with a manageable subset of columns and generate new favorable columns to enter the basis. The linear relaxation of MP over a subset of the variables is called the restricted master problem (RMP). The BP method first solves the RMP for a given set of columns using the simplex method. Then, dual variables  $\alpha_j$  and  $\gamma^i$  where  $i = 1, \dots, M + N$  and  $j = 1, \dots, M$ , associated with constraints (16) and (17) respectively, are generated and passed to the following pricing subproblem - called PP -

$$(PP) \quad \min_{p \in \Omega^i} \bar{c}_{ip} = c_{ip} - \sum_j \alpha_j a_{jp}^i - \gamma^i, \quad i = 1, \dots, N + M, \quad (19)$$

to price out favorable columns. Let  $\bar{c}_{ip}$  be the updated marginal cost for tail number  $i$  taking route  $p$ . If  $\bar{c}_{ip} \geq 0$  for all  $i = 1, \dots, N + M$ , then there is no column with a negative reduced cost and the current primal solution solves the RMP optimally. Otherwise, a subset of new columns is selected and added to the RMP. This completes one iteration and the enlarged RMP is re-optimized. The procedure continues until no column with negative reduced cost can be found.

These ideas are combined with a B&B scheme in the following branch-and-price algorithm.

*Step 1:* Determine an initial set of feasible columns, form the RMP, and go to Step 3.

*Step 2:* Choose a variable (or node) to branch.

*Step 3:* Solve the LP relaxation of the RMP and pass the dual variables to the PP.

*Step 4:* If columns with negative reduced costs in the PP exist,  
select a subset of those columns, add them to the RMP and go to Step 3.  
Otherwise, go to Step 5.

*Step 5:* If solution is integral, fathom. Otherwise, go to Step 2.

Branch-and-price approach has been used to solve large integer programs. For more information about this approach and implementation issues see [14] and [12].

## 5.4 Computational Performance of Enhanced Scheduling Methods

We test RESTORE, MIPS-CREW and BP on the same random data sets used for NETIP and MIPS in Section 4.3. We compare the solution quality obtained by these methods in terms of the total costs in Figure 3. The total computation time of these three methods is presented in Table 4, which includes both the setup time and solution time taken by OE to solve the problem. RESTORE uses MIPS in these experiments.

We solve BP to optimality, hence, the total cost of the schedule created by BP is the same as the one by MIPS-CREW. The total costs of the schedules obtained by RESTORE is compared to that of MIPS-CREW (or BP) in Figure 3. We only report the solution quality for high fleet sizes ( $N = 50, 70, 100$ ) and high load factors ( $l = 2, 3$ ). The values we report are average and maximum values over all the ten instances randomly created in each experiment. The cost savings (in percentage) achieved by MIPS-CREW and BP over RESTORE are represented in Figure 3. RESTORE can obtain feasible schedules that have costs 0.27% (on average) higher than the optimal solution. The maximum optimality gap for RESTORE is less than 1.1%.

MIPS-CREW generates optimal solutions at a cost of long computation time. MIPS-CREW outperforms BP in small size problems. However, when  $N \geq 50$  and  $l \geq 2$ , BP's computational advantage becomes visible. When  $(N, M) = (100, 300)$ , it takes around 22 minutes for BP to generate an optimal solution, which is nine times faster than MIPS-CREW. RESTORE is faster than both MIPS-CREW and BP.

In general, although MIPS-CREW creates optimal schedules, it is too slow to be a practical solution. RESTORE is a promising alternative for practical purposes in terms of simplicity, solution quality and computation time. However, it is not flexible to handle longer planning

horizons with multiple duty periods. BP is potentially the best alternative for solving large scale problems with high load factors and long planning horizons.

## 6 Conclusion and Future Research

We described a scheduling decision support tool, FlightOptimizer, for a major fractional management company (FMC) that operates fractional ownership aircraft. We first implemented a network flow model, called NETIP, for the aircraft scheduling problem. Because of its simplicity in implementation, NETIP has been used as the optimization engine in the company since Spring 2002. The average utilization of created schedules – a performance indicator used by senior executives – has increased to over 70%, compared to 62% before the implementation.

After the successful implementation of NETIP, the company asked for potential enhancements and alternative models for solving large instances of the aircraft scheduling problem and the combined aircraft and crew scheduling problem. Hence, we developed three alternative models and heuristics which are presented in this paper. The features of these models in terms of complexity in implementation, computation time, solution optimality, and flexibility in covering long planning horizon and incorporating complex operational rules, are summarized in Table 5. The problems are divided into four different categories. “SL” represents small size problems ( $N \leq 50$ ) with low load factors ( $l \leq 1$ ) and “LH” represents large size problems ( $50 \leq N \leq 100$ ) with high load factors ( $2 \leq l \leq 3$ ). The computation time (in CPU seconds) represents the average performance of each model using the minimal and maximal times in each category.

For the aircraft scheduling problem, MIPS works better than NETIP, especially for the problems in the “LH” category. Among the solutions for the combined aircraft and crew scheduling problem, MIPS-CREW provides optimal solutions (for a 24-hour horizon) but is too slow to be practical. On the other hand, RESTORE can generate schedules quickly with an average optimality gap of 0.27%. Thus, RESTORE is a preferred quick add-on solution for a 24-hour planning horizon. However, to solve large size problems covering longer planning horizons and to incorporate complicated operational rules, we believe that BP is a better alternative. With our flexible, module-based decision support tool, alternative optimization models can be easily plugged into FlightOptimizer to accommodate different operational requirements.

Several extensions to our model are under investigation to address other practical challenges faced by the FMC: (i) In fractional ownership programs both demand and aircraft supply change dynamically over time. For instance in the FMC around 25% of the requested trips arrive after the original schedule is created, and each day 10-12 aircraft require unplanned maintenance. Current schedule optimization is done by assuming that the demand and supply information is deterministic over the planning horizon. When there is a change on either side, the schedule has to be modified, but has to remain close to the original one because of several practical factors. A common complaint from managers when using optimization-based decision support tools is that small changes in input often translate into dramatically different solutions that sometimes are not managerially acceptable [2]. Therefore we aim to enhance FlightOptimizer so that it can update the schedule to accommodate disruptions in real time while keeping it as close to the original one as possible. (ii) All the models presented here are tested numerically for a 24-hour planning horizon. RESTORE cannot be adjusted in a straightforward way to solve problems with longer planning horizons that involve multiple duty periods. Alternative models are presented in [5] for multiple day horizons. However, they are MIP models and likely to have poor computation time with general purpose solvers. BP remains a promising solution with only minor complications in determining feasible routes. (iii) Current scheduling solutions consider all the trip requests as firm demand that must be satisfied by company aircraft or charter aircraft. However, it is possible that some demand

is optional, i.e., should be satisfied only if it is profitable. The FMC currently sees at least two sources of optional demand: Charter requests from other companies and special requests from customers in excess of their available flight hours. We believe that enhancing the current models to incorporate optional demands is an interesting and challenging extension.

## References

- [1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, and P.H. Vance. Crew scheduling. In R.W. Hall, editor, *Handbook of Transportation Science*, pages 493–521. Kluwer Scientific Publishers, 1999.
- [2] G. Brown, R. Dell, and R. Wood. Optimization and persistence. *Interfaces*, 27(5):15–37, 1997.
- [3] S. Carey. Ultimate upgrade: More fliers decide that 1st class just isn’t good enough. *The Wall Street Journal*, page 1, April 23 2002.
- [4] E. David. Outlook: Fractional ownership. *Business & Commercial Aviation*, 91(2):66, August 2002.
- [5] I. Karaesmen and P. Keskinocak. A note on scheduling time-shared aircraft: Alternative formulations for scheduling the aircraft subject to crew availability. Working paper, University of Maryland, College Park, MD, 2003.
- [6] P. Keskinocak. Corporate high flyers. *OR/MS Today*, December 1999.
- [7] P. Keskinocak and S. Tayur. Scheduling of time-shared jet aircraft. *Transportation Science*, 32(3):277–294, 1998.
- [8] J.L. Levere. Buying a share of a private jet. *The New York Times*, 1996.
- [9] C. Martin, D. Jones, and P. Keskinocak. Optimizing on-demand aircraft schedules for fractional aircraft operators. *Interfaces*, 33(3):22–35, September-October 2003.
- [10] National Business Aviation Association, Washington DC. *NBAA Business Aviation Factbook*, 2002.
- [11] D. Ronen. Scheduling charter aircraft. *Journal of Operational Research Society*, 51:258–262, 2000.
- [12] M.W.P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.
- [13] P. Sheridan. Some advantages of private aviation. *Pittsburgh Post Gazette*, January 21 2002.
- [14] P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(3):183–200, 1998.
- [15] A.L. Velocci, Jr. Fractional ownership programs finding new software essential. *Aviation Week & Space Technology*, June 1 2002.
- [16] A. Zagorin. Rent-a-jet cachet. *Time Magazine*, September 13 1999.

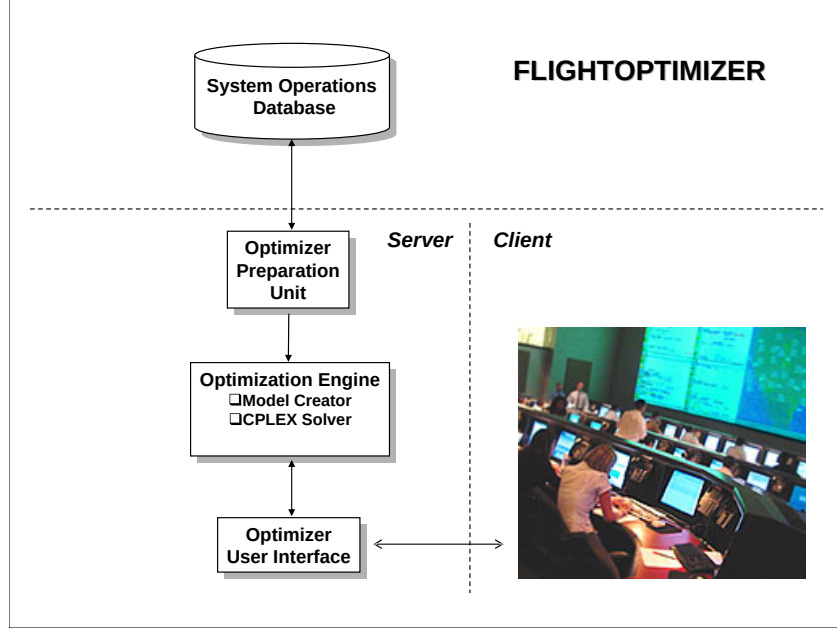


Figure 1: Architecture of the scheduling decision support tool “FlightOptimizer”

| Instance<br>(N,M) | Load Factor ( $l$ )<br>(M/N) | $\ AT\ $ | $\ ATT\ $ | Setup Time |      | Solution Time |      |
|-------------------|------------------------------|----------|-----------|------------|------|---------------|------|
|                   |                              |          |           | NETIP      | MIPS | NETIP         | MIPS |
| (40, 41)          | 1.02                         | 519      | 294       | 0.02       | 0.03 | 0.05          | 0.02 |
| (44, 70)          | 1.59                         | 1750     | 7330      | 0.19       | 0.58 | 1.16          | 0.42 |
| (51, 35)          | 0.69                         | 977      | 1949      | 0.06       | 0.15 | 0.14          | 0.09 |
| (55, 71)          | 1.29                         | 1354     | 2826      | 0.09       | 0.19 | 0.27          | 0.12 |
| (68, 135)         | 2.08                         | 4295     | 23386     | 1.45       | 6.44 | 9.50          | 3.25 |
| (72, 62)          | 0.86                         | 1564     | 4947      | 0.15       | 0.41 | 0.41          | 0.24 |
| (73, 90)          | 1.23                         | 3056     | 12195     | 0.45       | 1.72 | 1.34          | 0.74 |
| (74, 49)          | 0.66                         | 1441     | 3280      | 0.10       | 0.24 | 0.31          | 0.15 |
| (75, 105)         | 1.40                         | 3764     | 19474     | 0.97       | 4.33 | 5.17          | 3.66 |
| (76, 100)         | 1.32                         | 3683     | 11156     | 0.46       | 1.44 | 2.24          | 0.71 |

Table 1: Performance comparison of NETIP and MIPS on real data

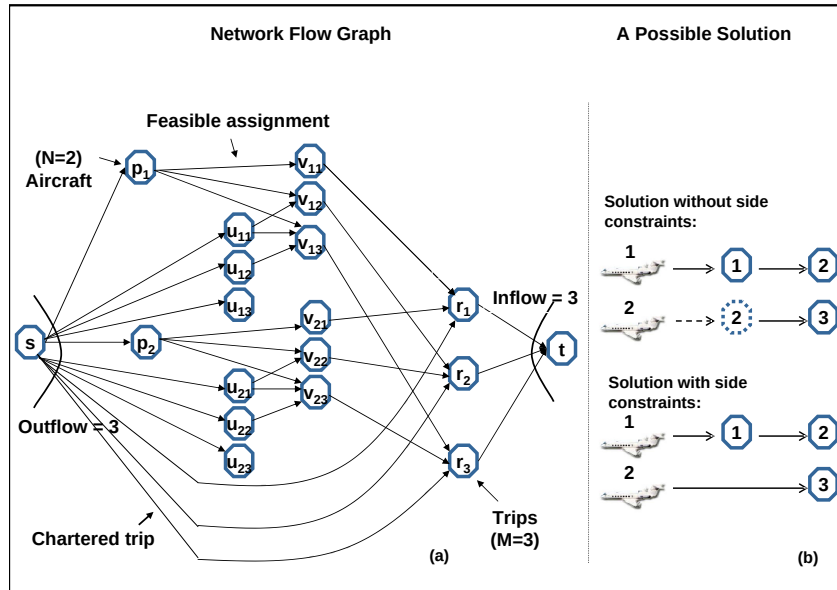


Figure 2: Illustration of the graph for the minimum cost network flow based formulation

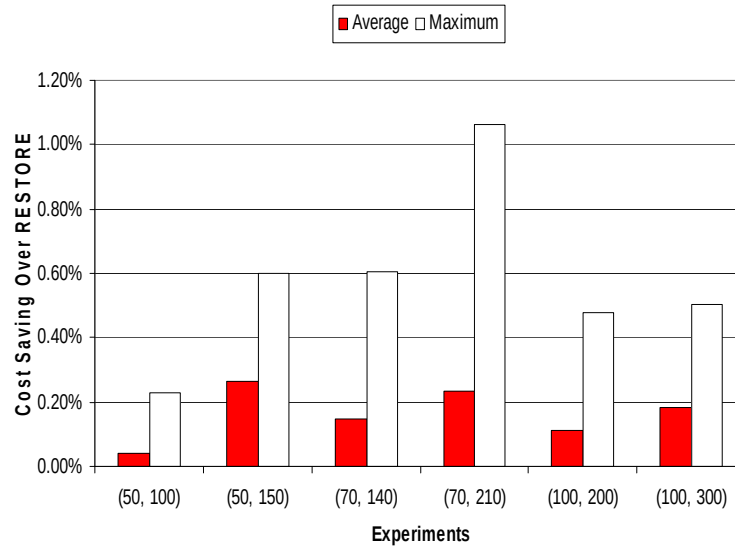


Figure 3: Cost saving by MIPS-CREW and BP compared to RESTORE

| Experiment<br>(N, M) | Load<br>Factor ( $l$ ) | $\ AT\ $ | $\ ATT\ $ | Setup Time |        | Solution Time |        | B&B Nodes |      |
|----------------------|------------------------|----------|-----------|------------|--------|---------------|--------|-----------|------|
|                      |                        |          |           | NETIP      | MIPS   | NETIP         | MIPS   | NETIP     | MIPS |
| (10, 10)             | 1                      | 43       | 20        | 0.01       | 0.01   | 0.01          | 0.01   | 0         | 0    |
| (10, 20)             | 2                      | 85       | 91        | 0.01       | 0.01   | 0.01          | 0.01   | 0         | 0    |
| (10, 30)             | 3                      | 133      | 195       | 0.01       | 0.02   | 0.02          | 0.01   | 0         | 0    |
| (30, 30)             | 1                      | 382      | 549       | 0.02       | 0.05   | 0.04          | 0.02   | 0         | 0    |
| (30, 60)             | 2                      | 741      | 2013      | 0.04       | 0.16   | 0.17          | 0.09   | 0         | 0    |
| (30, 90)             | 3                      | 1110     | 4315      | 0.08       | 0.36   | 0.95          | 0.34   | 0         | 0    |
| (50, 50)             | 1                      | 1006     | 2117      | 0.05       | 0.18   | 0.17          | 0.11   | 0         | 0    |
| (50, 100)            | 2                      | 2061     | 9093      | 0.22       | 1.17   | 2.93          | 1.08   | 1         | 1    |
| (50, 150)            | 3                      | 3121     | 21128     | 0.71       | 4.05   | 31.95         | 8.42   | 2         | 5    |
| (70, 70)             | 1                      | 2036     | 6181      | 0.17       | 0.71   | 0.59          | 0.35   | 0         | 0    |
| (70, 140)            | 2                      | 4074     | 25429     | 1.17       | 5.89   | 29.54         | 8.70   | 4         | 4    |
| (70, 210)            | 3                      | 6115     | 56023     | 4.07       | 18.82  | 246.21        | 55.28  | 27        | 14   |
| (100, 100)           | 1                      | 4091     | 18560     | 0.90       | 4.07   | 2.56          | 1.75   | 0         | 0    |
| (100, 200)           | 2                      | 8146     | 70823     | 6.82       | 28.79  | 212.07        | 46.48  | 12        | 2    |
| (100, 300)           | 3                      | 12274    | 164320    | 28.22      | 101.79 | 2154.82       | 868.58 | 64        | 177  |

Table 2: Performance comparison between NETIP and MIPS on randomly generated data

| Experiment<br>(N,M) | Total Computation Time |           |         |         |
|---------------------|------------------------|-----------|---------|---------|
|                     | RESTORE                | MIPS-CREW | BP(1%)  | BP      |
| (10, 10)            | 0.01                   | 0.01      | 0.61    | 0.61    |
| (10, 20)            | 0.02                   | 0.02      | 0.63    | 0.64    |
| (10, 30)            | 0.03                   | 0.05      | 0.65    | 0.65    |
| (30, 30)            | 0.07                   | 0.17      | 0.76    | 0.77    |
| (30, 60)            | 0.26                   | 0.97      | 1.40    | 1.40    |
| (30, 90)            | 0.70                   | 4.14      | 2.92    | 2.92    |
| (50, 50)            | 0.30                   | 1.86      | 1.61    | 1.61    |
| (50, 100)           | 2.25                   | 13.47     | 8.84    | 8.88    |
| (50, 150)           | 12.47                  | 77.12     | 27.12   | 27.27   |
| (70, 70)            | 1.06                   | 2.53      | 6.03    | 6.02    |
| (70, 140)           | 14.58                  | 63.09     | 46.32   | 47.21   |
| (70, 210)           | 74.10                  | 744.69    | 139.96  | 158.26  |
| (100, 100)          | 5.82                   | 21.87     | 28.14   | 28.17   |
| (100, 200)          | 75.27                  | 357.65    | 257.66  | 259.56  |
| (100, 300)          | 970.38                 | 12709.20  | 1177.66 | 1359.12 |

Table 3: Performance of RESTORE, MIPS-CREW and BP in randomly generated problems

| Experiment<br>(N,M) | Total Computation Time |           |         |
|---------------------|------------------------|-----------|---------|
|                     | RESTORE                | MIPS-CREW | BP      |
| (10, 10)            | 0.01                   | 0.01      | 0.61    |
| (10, 20)            | 0.02                   | 0.02      | 0.64    |
| (10, 30)            | 0.03                   | 0.05      | 0.65    |
| (30, 30)            | 0.07                   | 0.17      | 0.77    |
| (30, 60)            | 0.26                   | 0.97      | 1.40    |
| (30, 90)            | 0.70                   | 4.14      | 2.92    |
| (50, 50)            | 0.30                   | 1.86      | 1.61    |
| (50, 100)           | 2.25                   | 13.47     | 8.88    |
| (50, 150)           | 12.47                  | 77.12     | 27.27   |
| (70, 70)            | 1.06                   | 2.53      | 6.02    |
| (70, 140)           | 14.58                  | 63.09     | 47.21   |
| (70, 210)           | 74.10                  | 744.69    | 158.26  |
| (100, 100)          | 5.82                   | 21.87     | 28.17   |
| (100, 200)          | 75.27                  | 357.65    | 259.56  |
| (100, 300)          | 970.38                 | 12709.20  | 1359.12 |

Table 4: Average computation time of RESTORE, MIPS-CREW and BP to solve randomly generated problems

|                                        |    | Aircraft Scheduling |               | Combined Aircraft and Crew Scheduling |                |               |
|----------------------------------------|----|---------------------|---------------|---------------------------------------|----------------|---------------|
|                                        |    | NETIP               | MIPS          | RESTORE                               | MIPS-CREW      | BP            |
| Complexity in Implementation           |    | Low                 | Medium        | Medium                                | Medium         | High          |
| Solution Quality                       |    | Optimal             | Optimal       | Optimality gap range (0, 0.27%)       | Optimal        | Optimal       |
| Flexibility                            |    | Low                 | Low           | Low                                   | Low            | High          |
| Average Computation Time (CPU seconds) | SL | (0.02, 0.22)        | (0.02, 0.29)  | (0.02, 0.29)                          | (0.01, 1.86)   | (0.61, 1.61)  |
|                                        | SH | (0.02, 32.66)       | (0.02, 12.47) | (0.02, 12.47)                         | (0.02, 77.12)  | (0.64, 27.27) |
|                                        | LL | (0.22, 3.46)        | (0.29, 5.83)  | (0.29, 5.83)                          | (1.86, 21.87)  | (1.61, 28.17) |
|                                        | LH | (3.15, 2183)        | (2.25, 970)   | (2.25, 970)                           | (13.47, 12709) | (8.88, 1359)  |

Table 5: Feature summary of different models

# **Applications**

Abstracts



# SYSTEMS DESIGN FOR SCHEDULING

## *Open Tools*

Patrick De Causmaecker, Peter Demeester, Greet Vanden Berghe and Bart Verbeke

*Department Information Technology,*

*KaHo Sint-Lieven,*

*Gebr. Desmetstraat 1, 9000 Gent, Belgium*

*{Patrick.DeCausmaecker, Peter.Demeester, Greet.VandenBerghe, Bart.Verbeke}@kahosl.be*

Abstract:

## **1. PROBLEM DESCRIPTION**

We present an open system for (re)assigning people, whether or not belonging to the same department, in case of over –or understaffing.

In most companies employees are distributed over different departments. Experience teaches that when these departments become larger than a critical size, they become little islands that are reluctant to communicate with the outside world, which leads to suboptimal behaviour. In a police station, for example, policemen are assigned to several teams. It can occur that a certain team has extra workload, due to surveillance at an important sport event, while another team is overstaffed at the same time. This problem usually gets solved by relocating personnel temporarily, after negotiation between the responsible people of the teams. In this research, we investigate how to automate negotiation between departments [2].

## **2. AGENT MODEL AND LOCAL OPTIMISATION**

We developed a protocol to organise negotiation between different departments, which are represented as agents [3]. In this model a department is a set of people involved in one task. The number of tasks to be executed is thus equal to the number of departments. Employees with appropriate qualifications are initially assigned to these departments. Once the actual requirements are known, some departments may be overloaded while others

turn out to be overstaffed. The agent coordination model is designed to overcome this.

The initially assigned employees are scheduled using a tabu search algorithm. The evaluation of the constraints in this tabu search algorithm proceeds through the numbering method as described in [1]. The method offers a uniform evaluation procedure of linear complexity for a large class of constraints. From a developer's point of view, the advantage of this method is that the introduction of new constraints does not require developing new classes. The only thing that is needed is an expressive language that allows to describe the constraints. We use XML to represent constraints.

### 3. WEB SERVICES FOR DISTRIBUTED TIMETABLING

The distributed personnel scheduling application is built in such a way that every department is able to generate its own schedule. If necessary, the required calculations can be executed on a local computer of the department. The actual agent negotiation (based on the locally calculated schedule), however happens on a central computer. These negotiating agents have a direct connection with their counterparts on the local department computers. Every suggested relocation of personnel is evaluated locally and dependent on this evaluation, the proposed relocation will be accepted or not.

To make this application open, we implemented a web service that allows remote use of the application. This means that a remote client only has to know the correct URL of the web server where the application is deployed and the procedures (and arguments) that are needed to invoke the application. As a proof of concept we developed a GUI that communicates with the personnel scheduling application through these web services.

We present a prototype to study performance and implementation issues.

**Keywords:** distributed scheduling, agent technology, personnel rostering

### REFERENCES

- [1] E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe: *Fitness Evaluation for Nurse Scheduling Problems*, Proceedings of Congress on Evolutionary Computation, CEC2001, Seoul, IEEE Press, 2001, p. 1139-1146.
- [2] P. De Causmaecker, P. Demeester, G. Vanden Berghe, B. Verbeke: *Analysis of real-world personnel scheduling problems*, Proceedings of the 5<sup>th</sup> International Conference on Practice and Theory of Automated Timetabling, Pittsburgh, 2004, 183-197.
- [3] P. De Causmaecker, P. Demeester, G. Vanden Berghe, B. Verbeke: *A Coordination Model for Distributed Personnel Scheduling*, Proceedings of the 19<sup>th</sup> Orbel Conference, Louvain-La-Neuve, 2005.

# A DETERMINISTIC SCHEDULING APPROACH FOR WAFER FABRICATION FACILITIES

Scott J. Mason<sup>1</sup>, Michele Pfund<sup>2</sup>, and John Fowler<sup>2</sup>

*1-University of Arkansas, Fayetteville, AR, USA 2-Arizona State University, Tempe, AZ, USA*

**Abstract:** The primary goal of our proposed research effort is to develop solution methodologies for minimizing total weighted tardiness (TWT) for wafer fabrication facilities. We have taken the first steps towards this goal. In Mason et al. [1], we extended the classical job shop work of Pinedo and Singer [2] and developed a disjunctive graph formulation of the problem. Unlike previous job shop scheduling efforts, our disjunctive graph formulation accounts for batching tools, tool groups (identical machines operating in parallel), sequence-dependent setup times, and recirculating product flow. We provide some computational results that show the effectiveness of our approach.

**Key words:** Heuristic, Semiconductor Manufacturing, Total Weighted Tardiness.

## 1. OVERVIEW

Manufacturing integrated circuits on raw silicon wafers, also called (semiconductor manufacturing, is one of the most complex manufacturing processes in existence today. Semiconductor manufacturing consists of four major steps: 1) wafer fabrication, 2) sort or probe, 3) assembly/package, and 4) final test. Wafer fabrication is the most expensive and time consuming of these steps. In wafer fabrication, individual wafers of a given product (job) type are required to complete 300–400 process steps as they move through the wafer fabrication facility (wafer fab) in groups or lots of various size (for example, 25 wafers). Each lot of wafers competes with hundreds of other lots for processing time on the wafer fab's 50–100 different tool groups. The economic necessity to reduce capital spending dictates that such expensive machines be shared by all lots requiring the particular processing operation provided by the machine, even though they may be at different stages of their manufacturing cycle. These tool groups, which have machines that can range in price from \$100,000 to \$14,000,000, typically contain a number of identical machines operating in parallel, some of which may be dedicated to the production of one or more products. This results in a manufacturing environment that is different in several ways from both traditional flow shops as well as job shops (see Figure 1). The main consequence of the re-entrant flow nature is that wafers at different stages in their manufacturing cycle have to compete with each other for the same machines. The manner in which this competition is resolved has a clear impact on plant performance measures, particularly those related to meeting customer due dates.

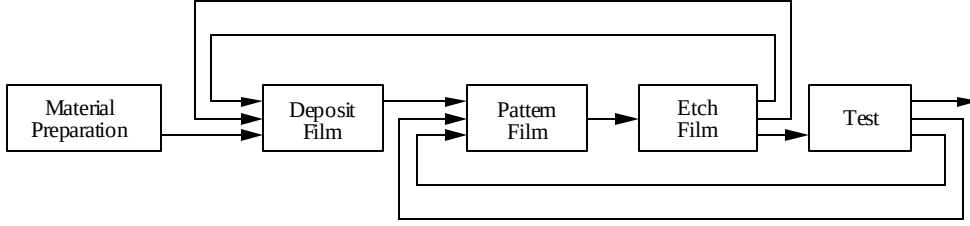


Figure 1. The wafer fabrication process (Re-entrant Flow)

In most wafer fabs today, the competition is resolved using dispatching techniques. In this presentation, we discuss our research efforts to develop deterministic scheduling solution methodologies for minimizing total weighted tardiness (TWT) for wafer fabrication facilities. We have taken the first steps towards this goal. In Mason *et al.* [1], we extended the classical job shop work of Pinedo and Singer [2] and developed a disjunctive graph formulation of the  $FJc \mid r_j, s_{jk}, B, recrc \mid \sum w_j T_j$  problem. Unlike previous job shop scheduling efforts, our disjunctive graph formulation accounts for batching tools, tool groups (identical machines operating in parallel), sequence-dependent setup times, and recirculating product flow. We provide an overview of our basic approach, briefly discuss the process of embedding our deterministic scheduling approach into a discrete event simulation, and present a comparison of our results with dispatching approaches on a variety of data sets.

## REFERENCES

- [1] Mason, S.J. , J.W. Fowler, and W.M. Carlyle, “A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops”, *Journal of Scheduling*, 5(3), 247-262(2002).
- [2] Pinedo, M.L. , and M. Singer, “A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop”, *Naval Research Logistics*, 46, 1-17 (1999).

# MULTIPLE ORDERS PER JOB SCHEDULING PROBLEMS

John Fowler, Erhan Kutanoglu, and Scott Mason

*Arizona State University, The University of Texas at Austin, The University of Arkansas*

**Abstract:** In this presentation, we introduce a new class of scheduling problems called Multiple Orders per Job Scheduling problems. In these problems, multiple orders (items) are combined into individual jobs (job formation) and then the jobs are scheduled. We provide an overview of this new class of problems and present issues that need to be addressed to solve these problems.

**Key words:** Multiple Orders Per Job, Semiconductor Manufacturing.

## 1. OVERVIEW

This paper introduces, models, and develops solution approaches for challenging problems found in manufacturing and logistics applications. At the core of these problems is the requirement to combine multiple entities, each with its own unique attributes, customers, and cost/quality/service sensitivities, into one or more containers for deployment or processing. The “filled” containers become *jobs* that must be scheduled efficiently throughout various segments of machines, factories, or supply chains to meet desired objectives. The resulting performance is a function of both entity-to-container assignment and container scheduling decisions. Considering that the scheduling-only portion is already a challenge for most problems, we are interested in problems that integrate job formation (multiple entities of different attributes are grouped to “form” jobs) and scheduling (jobs are scheduled in complex manufacturing and supply chain environments) as shown in Figure 1. We call these problems multiple orders per job scheduling (MOJS) problems. We are interested in MOJS problems in both manufacturing and logistics applications.

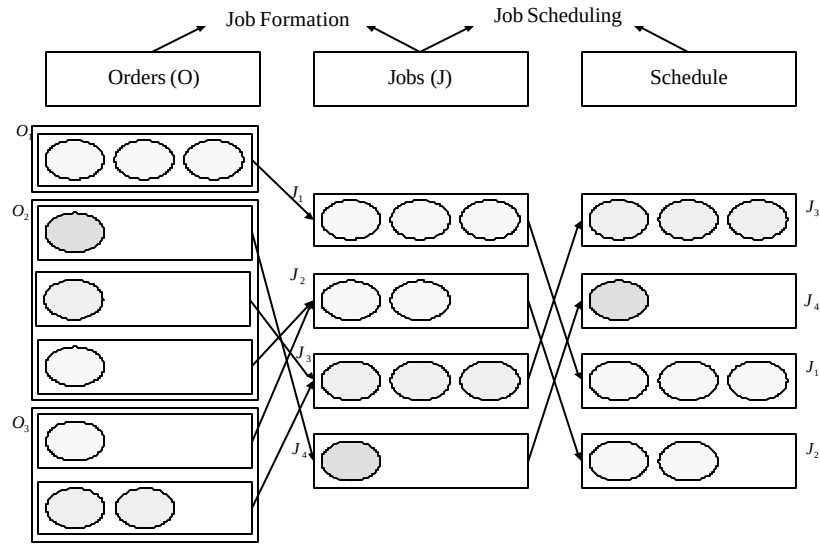


Figure 1 Multiple orders per job scheduling problems (patterns represent different product types)

# OPTIMAL REACTIVE SCHEDULING FOR JOB SHOP, MAKE-TO-ORDER INDUSTRIES: A MOULD MAKING INDUSTRY CASE STUDY

Marta C. Gomes<sup>1\*</sup>, Ana Barbosa-Póvoa<sup>2</sup>, Augusto Q. Novais<sup>3</sup>

<sup>1</sup>CESUR – Department of Civil Engineering and Architecture, <sup>2</sup>CEG-IST – Department of Engineering and Management), Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal.

<sup>3</sup>DMS – Department of Process Modelling and Simulation, Instituto Nacional de Engenharia, Tecnologia e Inovação, Est. Paço do Lumiar, 1649-038 Lisboa, Portugal.

\*Corresponding author. E-mail: marta.gomes@ist.utl.pt

Industrial environments are dynamic in nature and disturbances in the daily operation of the plant such as machine failures, processing time delays, arrival of new orders and unavailable material may cause a predetermined schedule to become inefficient or even infeasible. In real-time, the ability to rapidly react to such unexpected events and revise the schedule in an effective way (reactive scheduling) is thus as important as the scheduling problem itself.

Starting in the 1950s, a great deal of effort has been spent developing methods to generate optimal or near-optimal production schedules, yet interest in reactive scheduling techniques intensified only in the last decade.

In this paper a new approach for reactive scheduling of job shop, make-to-order industries is presented. A mixed-integer linear programming (MILP) model for job shop scheduling is derived with a view to its application in a dynamic context. Treatment of time is continuous, as opposed to discrete time models. The model is embedded in a reactive scheduling algorithm that shifts the scheduling horizon and solves the model iteratively to take account of changes in the system. One of the situations modelled is that of inserting new orders in the production schedule, an important and challenging problem in order-driven industries.

In the present approach the extent of the schedule changes may be controlled by the user with the definition of the subset of orders that may be rescheduled in each iteration of the algorithm, leaving the others unchanged. Additionally, further constraints may be added to the MILP model to limit changes in the operations previously scheduled. Solution continuity and system stability are thus preserved, an important issue in dynamic production environments where shop floor nervousness is undesirable.

The approach is illustrated by the solution of a mould making industry case study. This is an important industry in the context of the Portuguese economy. Mould makers operate in a job shop, make-to-order basis, producing one-of-a-kind products. Specific features of the mould making process, such as recirculation and assembly, are taken into account. Recirculation in a job shop occurs when certain jobs visit some machines more than once. In mould making, the process of carving a cavity has to be divided into stages due to the need of intermediate operations, hence implying recirculation. Also, moulds generally consist of a few large and other more numerous small parts, produced separately and assembled in the final stage of the process. Both recirculation and assembly, which have received scant treatment in the scheduling literature so far, are considered in the proposed MILP model.

# **Batch Scheduling**

Full Papers



# MULTIPLE ORDERS PER JOB BATCH SCHEDULING WITH INCOMPATIBLE JOBS

Vishnu Erramilli, Scott J. Mason

*Department of Industrial Engineering, 4207 Bell Engineering Center, University of  
Arkansas, Fayetteville, AR 72701, USA*

**Abstract:** The growth in demand for semiconductors has been accompanied by intense competition between semiconductor manufacturers to satisfy customer on-time delivery needs. This paper is motivated by scheduling the wafer fabrication phase of 300-mm semiconductor manufacturing, as it contains both batch processing operations and multiple customer orders being grouped into production jobs. The resulting *multiple orders per job (MOJ)* scheduling problem is presented for the batch-processing environment as exemplified by a diffusion oven. A mixed-integer programming formulation is presented for incompatible job families wherein only jobs that belong to the same family may be grouped together in a batch. This formulation is analyzed using an experimental design to minimize total weighted tardiness. While optimal solutions are attainable for small problem instances in a reasonable amount of computation time, heuristic development is warranted to support the analysis of larger, more practical MOJ batch scheduling problems. Promising heuristic approaches are identified that find near-optimal solutions very quickly.

**Key words:** semiconductor manufacturing, batching, total weighted tardiness

## 1. INTRODUCTION

The semiconductor manufacturing industry is one of the major industries in the world. Tremendous industrial growth has taken place over the last decade due to advancements in production technology and operational principles. The growth in demand for semiconductor products has been accompanied by intense competition between semiconductor manufacturers to satisfy customer needs. Satisfaction of customer needs is a very important part of any business. A very important dimension of customer satisfaction and goodwill is the on-time delivery of products. Indeed, the goal of most manufacturers is on-time delivery of high quality products while simultaneously reducing production costs.

On-time delivery of products can be promoted by reducing production cycle times. Cycle time reduction can be realized by purchasing and using extra equipment in the factory. However, tooling costs in the semiconductor manufacturing industry are very high, approaching \$15 million per tool; therefore, this option often is ruled out due to financial reasons. Alternately, production cycle times may be reduced more cost-effectively by efficiently planning and scheduling the existing resources in a semiconductor manufacturing facility (“wafer fab”). Planning and scheduling can lead to efficient utilization of production resources, which in turn can lower the time required to produce a product, promoting on-time delivery.

The typical semiconductor manufacturing process can be divided into four sub-systems/stages: wafer fabrication, wafer probe, wafer assembly, and wafer (final) testing (Uzsoy *et al.* 1992). Wafer fabrication and wafer probe are normally referred to as the “front-end” of the production process, while wafer assembly and final testing are commonly termed the “back-end.” In wafer fabrication the basic integrated circuit (IC) is fabricated, layer upon layer, by processes such as photolithography, etching, and diffusion. It is the most complex process/stage of production, containing between 300 and 500 process steps which are performed by various types of complex production tools. Individual circuits are electrically tested during wafer probe, with the functioning ICs being placed in various types of packages during wafer assembly. Finally, the completed, packaged ICs are tested once more during final testing before being sent to the customer.

This paper is motivated by the wafer fabrication and final testing phases of semiconductor manufacturing. In addition to a number of other processing complexities, such as re-entrant flow, parallel machines, and machine setup requirements, the wafer fabrication process is a complex job shop characterized by both discrete and batch processing tools (Mason *et al.* 2002). Discrete processors process individual wafers or entire “lots” of 25 wafers one at a time, while batch processors multiple lots of wafers simultaneously.

Batch processing is common in a number of manufacturing sectors. Jobs or the “lots” are aggregated and processed simultaneously to increase production efficiency on machines called batch processing machines. In most of these scenarios the decisions regarding the formation of the batches, i.e., which job to put in which batch and then the subsequent scheduling of these formed batches are crucial decisions to be taken. In some cases batch processing times are the longest therefore the batch processing machine is often the “bottleneck” of the manufacturing unit. In order to bring down the cycle time of the product it is really important to schedule such batch processing machines because it is essentially the effective control of the bottleneck which leads to effective control of the overall system.

## **2. LITERATURE REVIEW**

Scheduling of batches is a common part of scheduling of wafer fabs and other parts of the semiconductor manufacturing like the testing stage. Most batch scheduling literature usually considers the problem as consisting of two sub-problems, one of batch formation i.e., the assignment of the jobs to the batches and the other the sequencing of the formed batches on a single batch processing machine such as the diffusion chamber.

Ahmadi *et al.* (1992) consider batching and scheduling in a two-machine flow-shop consisting of different combinations of batch and discrete processors. The objectives investigated are to minimize the makespan and sum of job completion times by developing heuristics that run in polynomial time. The heuristics vary according to the system configuration. For example, a simple sorting based heuristic solves a flow-shop problem when the configuration is a batch processor followed by a discrete processor where as a dynamic programming approach solves it when the configuration is discrete processor followed by the batch processor.

Uzsoy (1994) schedules a single batch processing machine with non-identical job sizes to minimize total completion time and makespan. Uzsoy (1994) proves that the problems are NP-hard and heuristics based on the bin-packing problem are developed for minimizing the makespan. A branch and bound approach is used to solve the total completion time problem. Lower bound for the problem is calculated by relaxing the problem to be one of jobs with unit size. Heuristics based on greedy algorithms are developed for this problem and it is found that these heuristics work well.

### *Multiple Orders per Job Batch Scheduling with Incompatible Jobs*

Uzsoy (1995) extended the above problem to deal with batching and scheduling of jobs belonging to incompatible families. The problem is motivated by the diffusion process in the wafer fabrication process. Static jobs, where all jobs are available at the same time, are considered and it is shown that there should be no partial batches formed and using this greedy algorithm based heuristics are developed to minimize the makespan, maximum lateness, and total weighted completion time and these methods are extended to the parallel machine environment. Efficient heuristics are developed for the minimizing the makespan and maximum lateness in the presence of dynamic job arrivals.

Mehta and Uzsoy (1998) consider minimizing the total tardiness on a batch processing machine in which the jobs belong to incompatible families. The processing time of the jobs is processing time of the family to which it belongs. Mehta and Uzsoy (1998) prove that the problem is NP-hard when the number of families and batch capacity is arbitrary and then go on to describe some characteristics of the optimal solution. A dynamic programming algorithm is developed in which the initial batch is formed by using a greedy algorithm. Batches are added from the set of unscheduled batches in a way so as to minimize the tardiness function value. A heuristic is developed by modifying the ATC dispatching rule so that it is applicable to batches and this rule is called BATC. A batch priority index calculated for each batch and then the batch with the highest priority index is scheduled. A simpler decomposition heuristic is also developed in which batches are scheduled according to the BATC rule and then the dynamic program is applied to a subset to get optimal batching and scheduling and this is done till all jobs are covered. The heuristics were found to give near optimal solutions in reasonable amount of computation time.

Dobson and Nambimadom (2001) investigate the problem of batching and scheduling in which the jobs belong to different families and jobs have different sizes therefore use different amount of the resource, which is a single batching machine. The processing time of the batch is independent of the size and the number of jobs in the batch. The objective is to minimize the mean weighted flow time. An integer programming formulation is given which is found to be NP-hard. Heuristics are developed to solve the problem based on the structure of the problem. An LP relaxation of the integer program gives lower bounds to the problem. It is found that it is easier to break up the problem into two problems, one of efficiently packing the jobs into a batch and the second of sequencing the batches. According to the authors, once a sequence is obtained then it is easier to efficiently pack the jobs into batches. Three heuristics are developed: a greedy heuristic, a knapsack heuristic and a generalized assignment heuristic. From computational study it is seen that the assignment heuristic and knapsack heuristic are better than the greedy heuristic and produce near optimal solutions.

The above research is the closest to the research work in this paper in the sense that both look at batch processing of jobs belonging incompatible families with non-identical job sizes. There some fundamental differences which should be emphasized:

1. The overall environment in which the scheduling is being done is different in the sense that this paper deals with the multiple orders per job scheduling problem.
2. In Dobson and Nambimadom (2001), there are two basic units: jobs and batches and the decision of which batch belong to which family is made by the optimization model. In this paper there are 3 basic units: orders, jobs and batches and orders belong to certain families. The decision regarding assigning the orders to the jobs and the formed jobs to the batches is made by the optimization model.
3. In the paper, there is no restriction on the number of jobs in the system but in our models and MOJ environment, the number of jobs/FOUPS is limited.

### 3. INTEGRATED BATCH FORMATION AND SCHEDULING IN SEMICONDUCTOR MANUFACTURING

The semiconductor manufacturing industry has recently adopted a new standard as regards the size of the basic wafer on which the circuits are fabricated. The silicon wafers are now 300mm in diameter and the new fabs are called 300mm wafer fabs. Earlier the basic size was 200mm in diameter. The increase in size has brought about an increase in weight of each wafer. Now, complete automation of the material handling systems, both inter-bay and intra-bay, is required. Due to this increase in size the industry has modified most of its equipment and also the material handling system. The lots of wafers are now transferred in units called front opening unified pods (FOUPS). FOUFS are containers that hold a certain number of 300mm wafers in an inert nitrogen atmosphere which protects the wafers from contamination. FOUFS are expensive entities and it is not economical to assign each order its own FOUF. It would also overload the automated material handling system. Therefore 300mm semiconductor manufacturers would need to group orders from different customers into one FOUF.

Qu (2004) has investigated the problem of fitting orders belonging to different customers into the same jobs so that the scheduling is done at the job level but the customer satisfaction is measured at the order level. The problem investigated in this paper is the next step in this research of multiple orders per job scheduling problem. The methods and principles of grouping the orders into jobs and then the jobs into batches so as to schedule them on a single batch processing machine with capacity constraints are investigated so as to minimize a due-date related performance measure (total weighted tardiness) which is a true measure of on-time delivery of orders to customers.

Figure 1 gives a good representation of the problem of interest. Let  $O = \{1..N\}$  be the set of orders with  $N$  being the number of orders. Each order  $o \in O$  has the following parameters associated with it:

1.  $s_o$ , the size of order  $o \in O$  in number of wafers. The size of the order determines the number of slots it occupies in its associated FOUF.
2.  $w_o$ , the weight or priority of order  $o \in O$ . The manufacturer may assign each order its own importance or priority and this parameter quantifies the importance.
3.  $d_o$ , the due-date of order  $o \in O$ . This parameter represents the day that the order must complete manufacturing in order to meet the delivery date of the order.
4.  $f_o$ , the family of order  $o \in O$ . Each order belongs to a particular family of product types. There are a fixed number of families and each order belongs to one of these families.

There is a capacity limit on the number of wafers that can fit into a FOUF and it is denoted by  $K$ . The customer order size  $s_o$  can be less than, equal to or greater than this limit. The size and configuration of FOUFS is governed by an internationally recognized standard. Typical values for  $K$  are 13 or 25. For an order  $o$  with  $s_o < K$ , there is the potential for it to be combined with other orders. If  $s_o > K$ , then a single order cannot be placed into the FOUF due to capacity constraints. This, in turn, results in order splitting. An important basis of the research is that the orders are not going to be split up between FOUFS, therefore we assume that for all orders  $s_o < K$ .

Now, the FOUFS are later combined into batches which then undergo batch processing operations such as diffusion or burn-in operations. There is an upper limit on the number of FOUFS that can be placed into a batch and it is denoted by  $C$ . The objective investigated is that of the total weight tardiness (TWT) of the orders. It is computed as  $TWT = \sum_{o \in O} w_o T_o$ . The parameter  $T_o$ , the tardiness of order  $o$ , is

calculated as  $T_o = \max(0, \delta_o - d_o)$ . The objective perfectly captures the issue of on-time delivery of the orders to the customers which is a crucial requirement of any business organization. This paper focuses on the case of incompatible job families. In this environment, the orders belonging to certain

families have to be grouped together into jobs in such a way that ensures that orders belonging to different families are not grouped into the same job. Essentially it is like assigning a job to a particular family. The formed jobs then have to be grouped together into batches and this grouping should again ensure that orders belonging to different families are not assigned to the same batch. Again it is like assigning a batch to a particular family. This batching is done for the diffusion operation. The processing time of the batch is a constant independent of the contents of the batch. The families of orders have constant processing times which are known and the processing time of the batches is the processing time of the family it is assigned to.

#### 4. MIXED INTEGER PROGRAM

The following is a list of notation for the proposed optimization model:

|            |                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------|
| $K$        | Fixed capacity of the FOUP                                                                      |
| $C$        | Number of FOUPS/Jobs in a batch                                                                 |
| $\rho_f$   | Processing time of a single wafer of family type $f$ .                                          |
| $\delta_o$ | Completion time of order $o$ . The completion of the order is the completion time of the batch. |
| $\xi_b$    | Completion time of the batch $b$ and it depends on the environment.                             |
| $I_f$      | Index of orders $O$ which belong to family $f$ .                                                |
| $F_m$      | Set of Families                                                                                 |
| $B$        | Set of Batches                                                                                  |
| $N_B$      | Number of batches in the model = $F_m$                                                          |
| $F$        | Number of jobs in the system                                                                    |
| $O$        | Set of Orders, $\{1..N\}$                                                                       |
| $J$        | Set of Jobs, $\{1..F\}$                                                                         |

Parameters of the orders have been defined in the previous section. An important point to mention is that with out loss of generality, it is assumed for both models, the batches are processed in increasing order of the index, i.e., the sequence of processing the batches on the batching machines is known *a priori*.

The decision variables used in this model are as follows:

|           |                                                                       |
|-----------|-----------------------------------------------------------------------|
| $X_{ojb}$ | 1 if order $o$ is assigned to job $j$ is assigned to batch $b$ else 0 |
| $Y_{jf}$  | Integer, tracks which job $j$ belongs to which family $f$             |
| $Z_{bf}$  | Integer, tracks which batch $b$ belongs to which family $f$           |
| $L_{jb}$  | 1 if job $j$ is in batch $b$ else 0                                   |

The model formulation is as follows:

$$\begin{aligned}
 \text{Min } & \sum_{o \in O} w_o T_o \\
 \text{s.t. } & \sum_{o \in I_f} \sum_{b \in B} s_o X_{ojb} \leq K \quad \forall j \in J, f \in F & (1) \\
 & \sum_{j \in J} \sum_{b \in B} X_{ojb} = 1 \quad \forall o \in I_f, f \in F & (2) \\
 & Y_{jf} \geq X_{ojb} \quad \forall o \in I_f, j \in J, b \in B, f \in F & (3)
 \end{aligned}$$

$$\sum_j Y_{jf} \geq 1 \quad \forall f \in F \quad (4)$$

$$Z_{bf} \geq X_{ojb} \quad \forall o \in I_f, j \in J, b \in B, f \in F \quad (5)$$

$$\sum_b Z_{bf} \geq 1 \quad \forall f \in F \quad (6)$$

$$\sum_f Z_{bf} = 1 \quad \forall b \in B \quad (7)$$

$$L_{jb} \geq \sum_{o \in I_f} X_{ojb} / N \quad \forall j \in J, b \in B, f \in F \quad (8)$$

$$\sum_{b \in B} L_{jb} = 1 \quad \forall j \in J \quad (9)$$

$$\sum_{j \in J} L_{jb} \leq C \quad \forall b \in B \quad (10)$$

$$\xi_b \geq \rho_f Z_{bf} \quad \forall o \in I_f, j \in J, b \in B, f \in F \quad (11)$$

$$\xi_{b+1} \geq \xi_b + \rho_f Z_{b+1,f} \quad \forall j \in J, b \in [1 \dots N_{B-1}], f \in F \quad (12)$$

$$\delta_o \geq \xi_b - M \left( 1 - \sum_{j \in J} X_{ojb} \right) \quad \forall o \in I_f, b \in B, f \in F \quad (13)$$

$$T_o \geq \delta_o - d_o \quad \forall o \in I_f, f \in F \quad (14)$$

$$\delta_o \geq 0 \forall o \in O, \xi_b \geq 0 \forall b \in B$$

Constraints (1) and (2) are for the assignment of the orders to the jobs and to the batches. The first constraint ensures that the capacity of the job/FOUP is not exceeded by placing the orders into the jobs. The second constraint ensures that an order is assigned to a single job and a single batch and not split across the jobs and batches.

Constraints (3) and (4) are for ensuring that there is a family associated with each job, which depends on the orders present in the job. Due to capacity limitations, a single family may consist of more than a single job. Constraint (4) takes care of this issue.

Constraints (5), (6) and (7) are similar to the above constraints but they deal at the batch level to ensure that there is a unique family associated with each batch. Each batch can be associated with a single family and constraint (7) takes care of that. Constraints (8) and (9) describe the assigning of the jobs to the batches and together ensure that jobs are not split across batches. Constraint (10) is to check the fact that the capacity of the batch (described in terms of the number of jobs) is not exceeded.

The last 4 constraints (10-14) are for calculating the completion time of the batches, the completion time of the orders present in the batches and the tardiness of the orders. There is a constant processing time associated with each family and the processing time and completion time of the batch depends on the family it gets associated with and this depends on the association of the order with the families present in the batch.

## 5. EXPERIMENTAL DESIGN AND PRELIMINARY RESULTS

In order to investigate the performance of the proposed mixed integer programming (MIP) formulation of the multiple orders per job batch scheduling model, a number of experiments were

### Multiple Orders per Job Batch Scheduling with Incompatible Jobs

conducted using the experimental design described in Table 1. In the experiments, the penalty parameter big M was calculated as  $M = F * \max(\rho_f)$ . The numbers of FOUPS in the system, F, were calculated as

$$\max \left( \left\lceil \frac{F_m * N_F * \nu}{12\beta} \right\rceil + 1, F_m \right) \quad (15)$$

The weights in the instances were calculated using the discrete distribution *DU* [1, 15]. The above parameters were taken from Mason *et al.* (2004). The parameters related to due date calculations were taken from Mehta and Uzsoy (1998). Parameter *T* is the expected percentage of tardy jobs and *R* is the range parameter. A high value of *T* implies a lower mean due date and a high value of *R* means a higher variance among job due dates. For each experimental design factor combination, 10 problem instance replications were randomly generated and analyzed on a Pentium IV 3.00 GHz CPU with 1 GB RAM. The makespan used in Mehta and Uzsoy (1998) is calculated as:

$$C_{\max} = n_j \times m \times E[\rho_f] / B \quad (16)$$

where  $n_j$  = number of jobs/family,  $m$  = number of families,  $E[\rho_f]$  = expected processing time of families,  $B$  = batch capacity or number of jobs/batch. If we use a similar expression for our research problem, the expression that results is

$$C_{\max} = N_F \times F_m \times \text{avg}(\rho_f) / C \quad (17)$$

where the terms in the expression are from Table 1.

We note that for our MOJ batch scheduling research such a definition of makespan is inappropriate. This becomes clearer if we perform a dimensional analysis on the expression. In terms of base dimensions, the expression is

$$C_{\max} = \left( \frac{\#orders}{family} \right) \times (\# families) \times \left( \frac{time}{family} \right) / \frac{jobs}{batch} = \left( \frac{(\#order)(time)}{(job)} \right) \quad (18)$$

This is because the number of families equals the number of batches in our MOJ batch scheduling problem of interest. Therefore we see that the dimensions of makespan are not units of time. This difference occurs because; in this problem there are three elements (i.e., order, jobs, and batches) to be scheduled instead of two in the original problem (i.e., jobs and batches).

By looking at the expression we note that we need to multiply it by a  $(jobs/\#orders)$  term to get makespan in proper dimensional units. We note that this expression has an upper bound and lower bound based on the number of FOUPS in the system calculated in (15). The range for the makespan is:

$$\begin{aligned} C_{\max} &= \left[ \left( \frac{F_m}{N_F F_m} \right) \left( \frac{N_F F_m \bar{\rho}_f}{C} \right), \left( \frac{F_m N_F \nu / 12\beta}{F_m N_F} \right) \left( \frac{N_F F_m \bar{\rho}_f}{C} \right) \right] \\ &= \left[ \frac{F_m \bar{\rho}_f}{C}, \frac{N_F F_m \bar{\rho}_f \nu}{C(12\beta)} \right] \end{aligned} \quad (19)$$

Once initial experiments were conducted to verify the value of the percentage of tardy orders that resulted when *T* was set to 0.3 and 0.6, respectively, the resulting order tardiness values were too high, as over 70% of the jobs were tardy in the “loose” due date case of *T*=0.3. As order due date setting is a function of estimated makespan, a new expression was derived for makespan. For small problem

instances, the sum of order family processing times is a good estimate of the makespan, as the minimum number of batches used is equal to the number of families in the instance. As the number of batches required per family increases, future research is ongoing to obtain better estimates for makespan in larger problem instances, as simply using  $makespan = \sum_{f \in F_m} \rho_f$  can adversely underestimate due-date range and result in an unnecessarily tight order due date range.

The computation time required to obtain the optimal solutions for six-, eight-, and nine-order MOJ batch scheduling instances is given in Table 2. We note that these times seem reasonable on average, but worst case solution time can be quite poor for such small problem instances. Therefore, heuristics must be developed that give near optimal solutions in reasonable amount of computation time.

## 6. HEURISTIC DEVELOPMENT AND ANALYSIS

Mason *et al.* (2004) develop heuristics based on the first fit decreasing approach to solve the single machine multiple orders per job scheduling problem. The problem is treated similar to the bin-packing problem and first fit decreasing based heuristics, which is known to work well for the bin-packing problem, are applied to the single item and single lot problems to get solutions which are 1% and 2% above optimal respectively. A similar approach is used in heuristic development for the multiple orders per jobs batch scheduling problem with incompatible families. The problem can be considered to be consisting of sub-problems of assigning orders to jobs and then assigning these formed jobs to batches in order to minimize the total weighted tardiness of the orders. Mason *et al.* (2004) have developed good heuristics for the first sub-problem. The sub-problem that needs to be addressed is to assign these formed jobs to the batches while keeping the constraint of incompatible families satisfied. The heuristic development consists of:

- Order Sequencing
- Order to Job Assignment
- Job to Batch Assignment

### 6.1 Order Sequencing

As a preliminary step, orders are sorted according to two methods:

- EDD: Sort the orders in non-decreasing order of order due-dates.
- WEDD: Sort the orders in non-increasing ratio of weight and due-date of each order.

### 6.2 Order-to-Job Assignment

The next step is to assign these sorted orders into the FOUPS/jobs available. The order to job assignment is performed using (Mason *et al.* 2004):

- FFD: This is an implementation of the first-fit decreasing method in which the sorted orders are placed, one by one, into the jobs such that the capacity constraint of the jobs is not violated. Starting from the top of the order list, the selected order is put into the first job with available capacity. Once job 1 is full, then we start at the top and go through the order list to feasibly assign the available orders in job 2 in order to fill it as much as possible and continue till all the orders have been placed.
- FFD\_AJS: While orders are assigned to jobs according to first-fit criteria, this method tries to balance the size of the formed jobs. The job size is calculated as the sum of sizes of all orders



assigned to the job. At each iteration, the average size of each job is calculated by dividing the sum of sizes of all orders yet to be placed in the jobs by the number of currently unfilled jobs. This gives the expected average job size for each remaining job. The orders are assigned in such a way that they do not violate the FOUP capacity limit and also not exceed the expected average job size. If the size of the current job being filled exceeds the expected average job size or no more orders remain, then we stop filling the current job and consider the next job.

### 6.3 Job-to-Batch Assignment

These formed jobs are now assigned to the batches which are processed in increasing order of their indices respectively. Since decision making involves assigning the formed jobs to the available batches, *aggregated* (from the orders) information needs to be computed for the formed jobs. The aggregated information used for this purpose is:

- $w_j$  = Weight of job, which is the sum of weights orders placed in the job ( $w_j = \sum_{o \in j} w_o \forall j \in J$ )
- $d_j$  = Due-date of the job, which is computed to be minimum of due-dates of orders present in the job ( $d_j = \min(d_o) \forall o \in j, j \in J$ )
- $jpt_j$  = Job processing time. It is equal to the processing time of the family the job belongs to.

Using the aggregated information, the jobs are assigned to the batches using:

- FFDJB1: Sort jobs in increasing order of  $d_j$  and place in first available batch. Use weight of the jobs to break ties.
- FFDJB2: Sort jobs in decreasing order of ratio of weight of job ( $w_j$ ) times the number of orders in the job to sum of due-dates of the orders present in the job ( $\sum_{o \in J} d_o$ ) and place the jobs in the first available batch.
- FFDJBATC: The ATC rule (Vepsalainen and Morton, 1987) is applied to the jobs to sort them and then place these sorted jobs into the first available batch. ATC parameters are calculated as follows:

$$C_{\max} = \sum \rho_f \quad (20)$$

$$\bar{C}_{\max} = \frac{C_{\max}}{F_m} \quad (21)$$

$$R = \frac{\max(d_o) - \min(d_o)}{C_{\max}} \quad (22)$$

If  $R \leq 0.5$ ,  $K_l = 4.5 + R$ , else  $K_l = 6 - 2 * R$  (Pinedo, 2002). The ATC index is calculated according to Mehta and Uzsoy (1998):

$$I_j(t) = \frac{w_j}{jpt_j} \left( \exp \left( \frac{- \sum_{o \in J} \max(d_o - jpt_j - t, 0)}{K_l * \bar{C}_{\max}} \right) \right) \quad (23)$$

## Multiple Orders per Job Batch Scheduling with Incompatible Jobs

The parameter  $t$  is updated after each iteration to the job processing time ( $jpt_j$ ) of the recent most job placed in the batch. In the numerator, all the orders belonging to job  $j$  are considered one by one and their individual “slack” is summed up. The job with the highest index value is assigned to the first available batch. The process is repeated till all jobs have been assigned to the batches.

For each of the above methods, the constraint of incompatible families is implemented at all levels. It is ensured that orders belonging to different families are not assigned to the same job and the jobs belonging to different families are not assigned to the same batch. We note that there are  $2*2*3=12$  combinations to be run on each instance and the combination which consistently finds near optimal solutions in reasonable amount of computation time would be considered as being an applicable heuristic for this problem. The performance ratio of solutions obtained by the heuristics to the optimal solutions obtained using the MIP are calculated ( $PR = \frac{TWT(H)}{TWT(OPT^*)}$ ), where  $TWT(H)$  is the total weighted

tardiness value obtained by applying different combinations of simple heuristics  $H$  and  $TWT(OPT^*)$  is the solution obtained from the MIP. The different heuristic combinations are designated as shown in Table 3.

### 6.4 Experimental Results

From the selected  $PR$  results in Table 4, it can be seen that heuristic combinations  $H_3$ ,  $H_6$ ,  $H_9$ , and  $H_{12}$  give solutions that are 4% above optimal, on average. A given heuristic combination required less than one second to evaluate each of the problem instances under study. The other combinations in this table denote the worst case heuristic performance, while the missing heuristic combinations performed at a level between the best and worst cases. The average  $PR$  is the best when the number of orders per family ( $N_F = 4$ ) with the heuristic solution being 2% above optimal. At the time of this submission, the 12-order problem instance optimal solutions were not available. The  $PR$  is seen to increase to 4 % above optimal as we increase the number of families in the model from 2 to 3. The heuristics perform quite well for the factor level setting of  $T = 0.6$ , with the best heuristic producing a  $PR$  of nearly 1.0. However, the same heuristics give a 6% above optimal solution for parameter setting of  $T=0.3$ .

It is seen that combination of EDD, FFD1 and FFDBATC methods gave good results but the drawback of FFDBATC method lies in its sensitivity to the calculation of the look-ahead parameter. The look-ahead parameter was calculated using the order data in the instance. A grid search was performed to estimate appropriate values for the ATC rule’s look ahead parameter by running the combination  $H_3$  (the combination that gave the best heuristic solution) on the problem instances. Values of  $K_l = 2.5$  to  $K_l = 4$  all give solutions that are 3% above optimal, on average. When  $K_l = 3.5$ , the best performance results wherein average performance is 3.3 % above optimal and worst case performance is 5% above optimal.

## 7. CONCLUSIONS AND FUTURE WORK

A mixed-integer programming formulation for the multiple orders per job batch scheduling problem on a single batch processing machine is presented, as motivated by 300-mm semiconductor manufacturing diffusion operations wherein jobs that belong to the same family (product type) can be grouped together in a production batch. Experimental testing of this initial model formulation suggests

## Multiple Orders per Job Batch Scheduling with Incompatible Jobs

that while optimal solutions are achievable for this initial set of six to nine order problems within a reasonable amount of computation time, preliminary trends suggest the need for heuristic development to support the analysis of larger, more practical MOJ batch scheduling problems.

It can be inferred from the results that as we increase the number of families and the number of orders per family the computation time is increasing. Heuristics are developed that solve the problem in reasonable amount of computation time and give solutions that are near optimal. The heuristic approaches consist of sorting the orders according to a predetermined rule, placing these sorted orders in the available jobs and the placing these jobs in the available batches to minimize the total weighted tardiness of the orders. The combination of EDD (to sort the orders), FFD1 (to place the sorted orders into the jobs) and FFDBATC (to assign the formed jobs to batches) works efficiently to give solutions that are 3% (on-average) above optimal. The 12 orders results need to be incorporated in to the above obtained results to get a full set of results for the experiment.

In our future research efforts, we will be developing and testing a mixed integer programming formulation for the multiple orders per job batch scheduling problem with *compatible* batching, as well as extending this formulation to a multiple machine environment. These future research efforts will help us to continue along our path towards developing viable scheduling and dispatching approaches for MOJ environments as characterized by 300-mm semiconductor manufacturing.

## REFERENCES

- Ahmadi, J.H., Ahmadi, R.H., Dasu, S., Tang, C.S., 1992. Batching and scheduling jobs on batch and discrete processors, *Operations Research*, 40, 750-763.
- Chandru, V., Lee, C.Y., Uzsoy, R., 1993. Minimizing total completion time on batch processing machines. *International Journal of Production Research*, 31, 2097-2121.
- Devpura, A., Fowler, J.W., Carlyle, W.M., Perez, I., 2000. Minimizing total weighted tardiness on a single batch process machine with incompatible job families, *Symposium on Operations Research 2000*, Dresden, Germany, 366-371.
- Dobson, G., Nambimadom, R. S., 2001. The batch loading and scheduling problem, *Operations Research*, 49, 52-65.
- Kutanoglu, E., Tanrisever, F., Mason, S.J., 2004. Forming and scheduling jobs from multiple orders with different attributes: A computational study of special cases. *IERC conference 2004*, Houston, TX, USA.
- Knutson, K., Kempf, K., Fowler, J. W., Carlyle, M., 1999. Lot-to-order matching for a semiconductor assembly & test facility, *IIE Transactions*, 31, 1103-1111.
- Lee, C.Y., Uzsoy, R., Martin-Vega, L.A., 1992. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40, 764-775.
- Mason, S.J., Fowler, J. W. Carlyle, W. M., 2002. A modified shifting bottleneck heuristic for minimizing the total weighted tardiness, *Journal of Scheduling*, 5, 247-262.
- Mason, S.J., Qu, P., Kutanoglu, E., Fowler, J.W., 2004. The single machine multiple orders per job scheduling problem, *IIE Transactions*, being revised
- Mehta, S.V., Uzsoy, R., 1998. Minimizing total tardiness on batch processing machine with incompatible job families, *IIE Transactions*, 30, 165-178.
- Pinedo, M., 2002, *Scheduling Theory, Algorithms, and Systems*, Prentice Hall Inc., Upper Saddle River, New Jersey.
- Qu, P., 2004, *The Multiple orders per job scheduling problems*, PhD Dissertation, University of Arkansas, Fayetteville, AR.
- Uzsoy, R., Lee, C.Y., Martin-Vega, L.A. 1992. A review of production planning and scheduling models in the semiconductor industry. Part I: system characteristics, performance evaluation and production planning, *IIE Transactions*, 24, 47-59.
- Uzsoy, R., 1994. Scheduling a single batch processing machine with nonidentical job sizes, *International Journal of Production Research*, 32, 1615-1635.
- Uzsoy, R., 1995. Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33, 2685-2708.

*Multiple Orders per Job Batch Scheduling with Incompatible Jobs*

Vepsalainen, A.P.J., Morton, T.E., 1987. Priority rules for job shops with weighted tardiness costs, *Management Science*, 33(8), 1035-1047.

*Multiple Orders per Job Batch Scheduling with Incompatible Jobs*

Table 1. Experimental Design

| Factors                                              | Levels | Level Description                                                                                                                                                                                                                      |
|------------------------------------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Problem Type ( $P$ )                                 | 1      | Incompatible Job Families                                                                                                                                                                                                              |
| Number of Families( $F_m$ )                          | 2      | 2,3                                                                                                                                                                                                                                    |
| Order Size ( $s_o$ )                                 | 2      | Discrete uniform $DU\left[\nu - \frac{\nu+1}{2}, \nu + \frac{\nu+1}{2}\right]$ , where $\nu \in \{3,5\}$                                                                                                                               |
| FOUP Capacity ( $K$ )                                | 2      | $12\beta + 1$ , where $\beta \in \{1,2\}$                                                                                                                                                                                              |
| Capacity of Batches( $C$ in terms of number of jobs) | 2      | 2,3                                                                                                                                                                                                                                    |
| Number of Orders( $N_F$ ) per family                 | 2      | 3,4                                                                                                                                                                                                                                    |
| Due Date ( $d_o$ )                                   | 4      | $DU\left[\mu - \frac{\mu R}{2}, \mu + \frac{\mu R}{2}\right]$ , where $\mu = \text{makespan} * (1 - T)$<br>Makespan = $\sum \rho_f$<br>$\rho_f$ = processing times of the families in the instance<br>$T = 0.3, 0.6$<br>$R = 0.5, 2.5$ |
| Processing Time per family ( $\rho_f$ )              | 1      | 2 with probability of 0.2<br>4 with probability of 0.2<br>10 with probability of 0.3<br>16 with probability of 0.2<br>20 with probability of 0.1                                                                                       |

96 Unique Design Factor Combinations  
10 Problems per Design Factor Combination  
**960 Total Problems Instances Investigated**

*Multiple Orders per Job Batch Scheduling with Incompatible Jobs*

Table 2. Computation Time for MIP

|                             |                   | Time(secs) | Min | Max   |
|-----------------------------|-------------------|------------|-----|-------|
| Order Size( $\nu$ )         | (1,3,*,*,*,*,*)   | 4.5        | 0.0 | 55.2  |
|                             | (1,5,*,*,*,*,*)   | 46.3       | 0.1 | 805.4 |
| Size Factor( $\beta$ )      | (1,*,1,*,*,*,*)   | 49.9       | 0.1 | 805.4 |
|                             | (1,*,2,*,*,*,*)   | 0.8        | 0.0 | 3.5   |
| Number of Families( $F_m$ ) | (1,*,*,2,*,*,*)   | 4.0        | 0.0 | 187.8 |
|                             | (1,*,*,3,*,*,*)   | 68.1       | 0.6 | 805.4 |
| Orders per Family( $N_F$ )  | (1,*,*,*,3,*,*)   | 34.4       | 0.0 | 805.4 |
|                             | (1,*,*,*,4,*,*)   | 7.4        | 0.0 | 137.8 |
| Batch Capacity( $C$ )       | (1,*,*,*,*,2,*)   | 25.2       | 0.0 | 805.4 |
|                             | (1,*,*,*,*,3,*)   | 25.5       | 0.0 | 739.7 |
| % Tardy( $T$ )              | (1,*,*,*,*,0.3,*) | 16.9       | 0.0 | 524.6 |
|                             | (1,*,*,*,*,0.5,*) | 33.9       | 0.0 | 805.4 |
| Due Date Range( $R$ )       | (1,*,*,*,*,*,0.5) | 29.9       | 0.0 | 805.4 |
|                             | (1,*,*,*,*,*,2.5) | 20.8       | 0.0 | 677.8 |
| Overall                     | (1,*,*,*,*,*,*)   | 25.4       | 0.0 | 805.4 |

*Table 3. Heuristic Combinations for Evaluation*

| Designation | Sort Order | Order to Job | Job to Batch |
|-------------|------------|--------------|--------------|
| $H_1$       | EDD        | FFD          | FFDJB1       |
| $H_2$       | EDD        | FFD          | FFDJB2       |
| $H_3$       | EDD        | FFD          | FFDJBATC     |
| $H_4$       | EDD        | FFD_AJS      | FFDJB1       |
| $H_5$       | EDD        | FFD_AJS      | FFDJB2       |
| $H_6$       | EDD        | FFD_AJS      | FFDJBATC     |
| $H_7$       | WEDD       | FFD          | FFDJB1       |
| $H_8$       | WEDD       | FFD          | FFDJB2       |
| $H_9$       | WEDD       | FFD          | FFDJBATC     |
| $H_{10}$    | WEDD       | FFD_AJS      | FFDJB1       |
| $H_{11}$    | WEDD       | FFD_AJS      | FFDJB2       |
| $H_{12}$    | WEDD       | FFD_AJS      | FFDJBATC     |

Table 4. Best and Worst Case Heuristic  $PR$  Results

|                              | $PR$  |       |       |       |       |          |          |
|------------------------------|-------|-------|-------|-------|-------|----------|----------|
|                              | $H_3$ | $H_4$ | $H_6$ | $H_7$ | $H_9$ | $H_{10}$ | $H_{12}$ |
| Order Size ( $\nu$ )         |       |       |       |       |       |          |          |
| (1,3,*,*,*,*,*)              | 1.029 | 1.347 | 1.033 | 1.336 | 1.029 | 1.341    | 1.033    |
| (1,5,*,*,*,*,*)              | 1.044 | 1.409 | 1.048 | 1.377 | 1.044 | 1.406    | 1.047    |
| Size Factor ( $\beta$ )      |       |       |       |       |       |          |          |
| (1,*,1,*,*,*,*)              | 1.039 | 1.376 | 1.045 | 1.35  | 1.039 | 1.369    | 1.044    |
| (1,*,2,*,*,*,*)              | 1.034 | 1.379 | 1.036 | 1.363 | 1.034 | 1.378    | 1.036    |
| Number of Families ( $F_m$ ) |       |       |       |       |       |          |          |
| (1,*,*,2,*,*,*)              | 1.031 | 1.335 | 1.035 | 1.302 | 1.031 | 1.331    | 1.034    |
| (1,*,*,3,*,*,*)              | 1.048 | 1.463 | 1.051 | 1.464 | 1.048 | 1.459    | 1.051    |
| Orders Per Family ( $N_F$ )  |       |       |       |       |       |          |          |
| (1,*,*,*,3,*,*)              | 1.042 | 1.386 | 1.046 | 1.380 | 1.042 | 1.381    | 1.045    |
| (1,*,*,*,4,*,*)              | 1.026 | 1.361 | 1.030 | 1.310 | 1.026 | 1.360    | 1.029    |
| Batch Capacity ( $C$ )       |       |       |       |       |       |          |          |
| (1,*,*,*,*,2,*,*)            | 1.046 | 1.368 | 1.050 | 1.364 | 1.046 | 1.365    | 1.049    |
| (1,*,*,*,*,*,3,*,*)          | 1.027 | 1.388 | 1.031 | 1.348 | 1.027 | 1.382    | 1.031    |
| % Tardy ( $T$ )              |       |       |       |       |       |          |          |
| (1,*,*,*,*,*,0.3,*)          | 1.064 | 1.410 | 1.067 | 1.379 | 1.064 | 1.408    | 1.067    |
| (1,*,*,*,*,*,0.6,*)          | 1.009 | 1.346 | 1.014 | 1.333 | 1.010 | 1.340    | 1.013    |
| Range ( $R$ )                |       |       |       |       |       |          |          |
| (1,*,*,*,*,*,*,0.5)          | 1.030 | 1.380 | 1.036 | 1.364 | 1.030 | 1.373    | 1.035    |
| (1,*,*,*,*,*,*,2.5)          | 1.044 | 1.376 | 1.045 | 1.349 | 1.044 | 1.375    | 1.045    |
| Overall                      |       |       |       |       |       |          |          |
| (1,*,*,*,*,*,*,*)            | 1.037 | 1.378 | 1.040 | 1.356 | 1.037 | 1.374    | 1.040    |



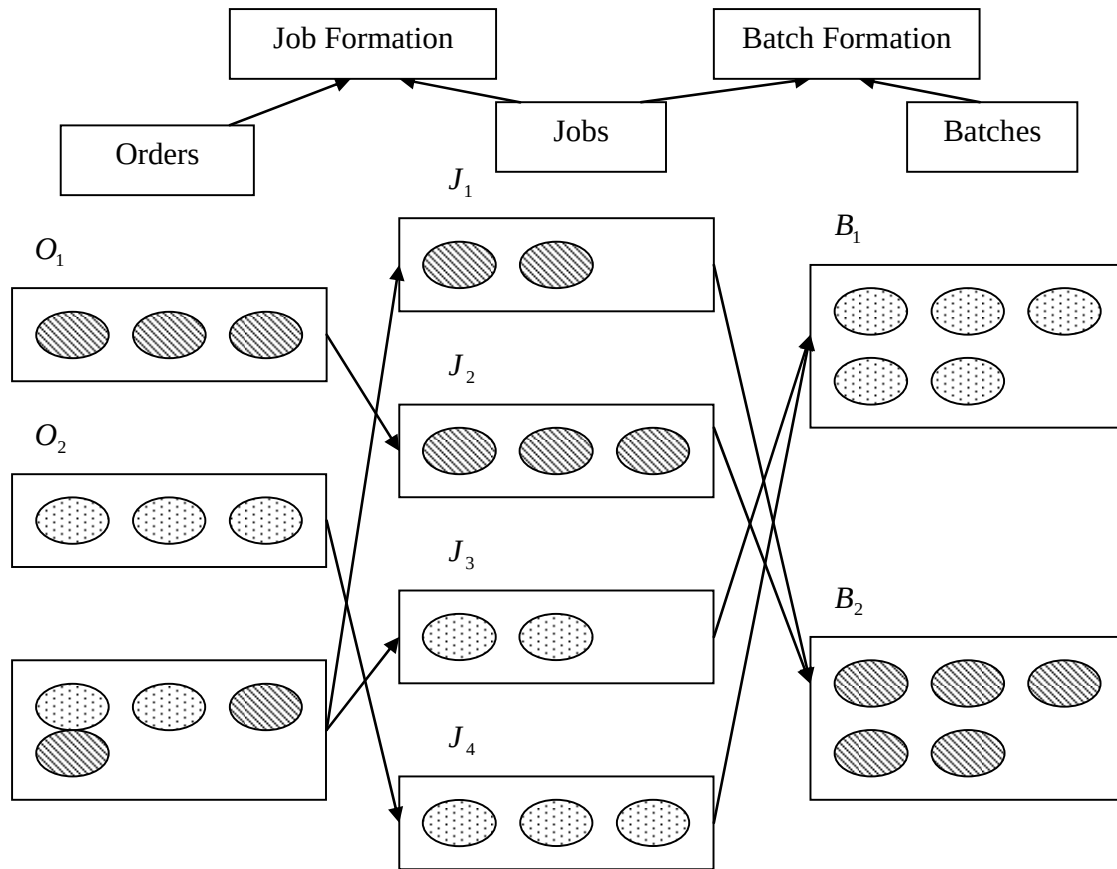


Figure 1. The multiple orders per job batch scheduling problem

# **Batch Scheduling**

Abstracts

# A DECOMPOSITION APPROACH FOR BATCH SCHEDULING IN THE PROCESS INDUSTRY

Christos T. Maravelias

*Department of Chemical & Biological Engineering, Univeristy of Wisconsin  
Madison, WI 53706*

The scheduling of batch plants in the process industry is an important and challenging problem for both academia and practice. Despite the improvements in computer hardware and optimization software many batch scheduling problems remain hard to solve, mainly due to the structural complexity of the production facilities and the fact that the type and number of tasks (jobs) as well as the assignment of tasks to equipment units (machines) are optimization decisions.

In this work, we first show that most batch scheduling problems can, in principle, be decomposed into a master *assignment* subproblem and a slave *sequencing* subproblem. This is particularly helpful because most of the *sequencing* subproblems have been extensively studied in the OR community and can be effectively solved. We propose, therefore, a decomposition approach, where the original problem is decomposed into an aggregate master MIP subproblem and a slave subproblem that can be solved using various algorithms (e.g. Constraint Programming, heuristics, problem-specific scheduling algorithms). In the master MIP problem we determine the type and number of tasks to be carried out as well as the assignment of tasks to units. For a given assignment (i.e. fixed number and type of tasks and task-unit assignments), in the subproblem we check feasibility and solve the sequencing problem. The slave subproblem is also used to generate Benders cuts that are added in the cut-pool of the master subproblem. The proposed framework can be implemented either as an iterative scheme or as a branch-and-cut algorithm. The advantage of the proposed scheme is that it combines the complementary strengths of different solution methods. We use Mathematical Programming to optimize (i.e. identify partially good solutions), and specialized algorithms to check feasibility and solve the sequencing subproblem. The proposed decomposition approach, however, has the following two limitations:

1. It is difficult to develop “tight” formulations for the MIP master subproblem. This implies that a lot of computational time can be potentially spent for the examination of infeasible assignments.
2. It is difficult to generate strong Benders (integer) cuts, because the “source” of infeasibility cannot always be identified. The generated Benders cuts, therefore, cut-off a very small region.

To address the first limitation we develop pre-processing algorithms, while for the second one we study sub-networks of the original problem and use the solution (or the solution process) of the slave subproblem. Pre-processing algorithms are effective in two ways: a) tighten the time window of assignment constraints of the master subproblem, and b) identify infeasible assignments (that are feasible for the original master problem) and a priori generate integer cuts to exclude them. Moreover, the decomposition of the production network into subnetworks enables us to develop strong integer cuts with few integer variables. Ideally, we want to generate cuts that include only the variables that are “responsible” for an infeasible solution.

The proposed approach was implemented in three classes of problems with very good computational results (more details and preliminary results can be found in Maravelias and Grossmann (2004a, 2004b)):

1. Single-stage batch plants: The one-machine algorithm of Balas and Vazacopoulos (1995) is used for the solution of the subproblem. Pre-processing is used for the a priori generation of integer cuts. Both the minimization of makespan and the minimization of cost (with release and due times) are considered.
2. Multi-stage batch plants: The Shifting Bottleneck Procedure (SBP) of Balas et al. (1998) is used for the solution of the slave subproblem. The SBP provides us with information that can be used for the derivation of strong integer cuts. Pre-processing is also used for the generation of integer cuts.
3. Multi-purpose batch plants: The sub-problem is solved using Constraint Programming methods. A network-based pre-processing algorithm is used to tighten the MIP master problem and different types of integer cuts are developed.

Balas, E.; Lenstra, J. K.; Vazacopoulos, A. The One Machine Problem with Delayed Precedence Constraints and its Use in Job-shop Scheduling. *Management Science*, **1995**, 41, 94 – 109.

Balas, E.; Vazacopoulos, A. Guided Local Search with Shifting Bottleneck for Job Shop Scheduling, *Management Science*, **1998**, 44(2), 262-275.

Maravelias, T.C., Grossmann, I.E. A Hybrid MIP/CP Decomposition Approach for the Short Term Scheduling of Multipurpose Plants. *Computers and Chemical Engineering*, **2004**, 28, 1921-1949.

Maravelias, C.T; Grossmann, I.E. Using MILP and CP for the Scheduling of Batch Chemical Processes. *Lect Notes Comput. Sc.*, **2004**, 3011: 1-20.

# **Delivery Scheduling**

Abstracts

# How to maximize rewards in an incentivised made-to-order scheduling environment

Celia A. Glass <sup>\*</sup>*Cass Business School, City University, London, and*  
(Daniel) S.T. Ng <sup>†</sup>*Hong Kong Polytechnic University.*

28 March 2005

In a high demand competitive business, providers may be selective about the orders which they accept and purchasers shop around for a suitable service provider. This paper considers the effect of incentivising earlier delivery dates, on the market's ability to match up supply and demand. It focuses on an individual supplier, and their strategy for selecting orders and allocating limited resources to reduce the processing time. The context is one of controllable processing times, and multi-item orders, with items falling into range of types, each of which are handled (i.e. produced or serviced) consecutively with a set up time between. Customers have no interest in orders which arrive late. This paper presents a method for selecting orders and targetting reduction in processing times, so as to maximise the reward for an individual supplier. The extent to which such a policy is of mutual benefit to potential customers is discussed.

This paper generalises results for a single machine assembly problem, with families of items. Set up times are sequence independent, and there is item-availability within in a batch of jobs of the same family. An order is completed only when all its items are ready. Process times are controllable at a cost per order proportional to its total processing time, within a bound on total compression of processing time. Profit arises from those orders accepted, with no penalty for orders rejected by the supplier. The net profit on an early order is given by the initial profit from the customer less the cost of compressing the order's processing time. The objective function is that of total net profit number of early jobs. A Dynamic Programme is developed to solve this problem to optimality, and its computational complexity analysed. While the presentation is in terms of a particular objective function and type of controllable processing times, the method is shown to generalise in both respects.

**Keywords:** Scheduling: controllable processing times, single machine, multi-operation jobs, dynamic programming

Supply Chain Management: multi-part orders, incentivised delivery times, multiple purchasers, single supplier.

---

<sup>\*</sup>C.A.Glass@city.ac.uk, Cass Business School, City University, London, EC1Y 8TZ,U.K

<sup>†</sup>Daniel.Ng@inet.polyu.edu.hk, Department of Logistics, the Hong Kong Polytechnic University, Hong Hum, Kowloon, Hong Kong.

# On the Capacitated Packing-Delivering Problem with a Fixed Route

Lei Lei, Rutgers Business School, Rutgers University, [llei@andromeda.rutgers.edu](mailto:llei@andromeda.rutgers.edu)

Sunju Park, School of Business, Yonsei University, Korea, [boxenju@yonsei.ac.kr](mailto:boxenju@yonsei.ac.kr)

\*Shuguang Liu, School of Business, State University of New York at New Paltz, [liush@newpaltz.edu](mailto:liush@newpaltz.edu)

## ABSTRACT

The packing-delivering problem is concerned with packing products with different capacity requirements (e.g., cubic feet or weight) into capacitated vehicles and scheduling the vehicle delivery operations so that a given objective is optimized while the customer demands and the vehicle capacity constraints are satisfied. In this study, we solve a variation of this problem, which involves  $K$  distinct products,  $K \geq 1$ , and a single capacitated vehicle with a fixed sequence of customer locations to be visited. The vehicle capacity is restricted by the total loading weight, and each customer may order multiple products with different quantity for each product. The objective is to minimize the total time needed to deliver to all the customers. Let  $\mathbf{P}$  denote this problem variation.

We prove that  $\mathbf{P}$  is strongly polynomial time solvable if each customer can be visited only once, but becomes NP-hard in strong sense if customers accept split deliveries. For the later case, the differences between  $\mathbf{P}$  and the known single-product split delivery problem are discussed, and the new problem properties are presented. We then analyze the conditions of optimality and show that the results from this analysis can greatly reduce the search effort without affecting the solution quality. A local optimization algorithm with a computational complexity of  $O(K \cdot n^3 \log(n))$  is proposed for the case that allows split deliveries and evaluated through 14,000 randomly generated test problems with a wide range of parameter values and product types. A consistent performance improvement achieved by the proposed local optimization algorithm over that by the optimal non-split packing-delivery schedules is observed.

Keywords: delivery scheduling, bin-packing, split delivery, fixed route

---

\*Corresponding author

# **Evolutionary Algorithms**

Abstracts



# **Ant Colony Optimization for Power Plant Maintenance Scheduling Optimization: A Simplified Hydro Tasmania System**

Wai Kuan Foong

Centre for Applied Modeling in Water Engineering (CAMWE), School of Civil & Environmental Engineering, The University of Adelaide, Australia 5005.

+61 40 3162457

[wfoong@civeng.adelaide.edu.au](mailto:wfoong@civeng.adelaide.edu.au)

Holger R. Maier

Centre for Applied Modeling in Water Engineering (CAMWE), School of Civil & Environmental Engineering, The University of Adelaide, Australia 5005.

+61 8 83034139

[hmaier@civeng.adelaide.edu.au](mailto:hmaier@civeng.adelaide.edu.au)

Angus R. Simpson

Centre for Applied Modeling in Water Engineering (CAMWE), School of Civil & Environmental Engineering, The University of Adelaide, Australia 5005

+61 8 83035135

[asimpson@civeng.adelaide.edu.au](mailto:asimpson@civeng.adelaide.edu.au)

## **Abstract**

In order to maintain a reliable and economic electric power supply, the maintenance of power plants is becoming increasingly important. In this paper, a formulation that enables ant colony optimization (ACO) algorithms to be applied to the power plant maintenance scheduling optimization (PPMSO) problem is developed and tested on a simplified hydro system of Hydro Tasmania, Australia. In the PPMSO problem, there are a number of constraints to be satisfied when scheduling for maintenance, and whether these constraints can be met depends on some external conditions, such as the inflow into reservoirs and system demand. In order to obtain optimal or near-optimal schedules under these stochastic external conditions, the ACO formulation developed is capable of incorporating options of deferring outages, as well as shortening of outage durations. Heuristic formulations for favoring normal outage duration over deferral and shortening options are also developed. In this paper, the above formulation was applied to a simplified version of the Hydro Tasmania system under three different storage inflow conditions, namely dry, intermediate and wet years. The simplified system consists of the largest power station in Tasmania and a cascade system that encompasses four power stations. There are 14 maintenance tasks to be scheduled over a 12-month horizon, which includes a 42-day major upgrade of the largest power station. The PPMSO problem is therefore defined as the determination of an optimal maintenance schedule such that the maintenance tasks accomplished and the Energy In Storage (EIS) of the system are maximized, subject to a number of constraints, which include load constraints, as well as precedence and timeframe constraints, for investigative and actual maintenance tasks. The outcome of the ACO runs suggests that the major upgrade maintenance can proceed as normal in a wet year, but in an intermediate year and dry year, the outage duration needs to be shortened from 42 days to 14 days and 8 days, respectively. Furthermore, the EIS of the system is observed to be increasing throughout the optimization process under all three inflow conditions. In conclusion, the results obtained by using the ACO formulation developed are promising and the formulation will be further investigated and applied to the real Hydro Tasmania system.

# **Job-shop Scheduling**

Full Papers

# AN APPROACH TO PREDICTIVE-REACTIVE SCHEDULING OF PARALLEL MACHINES SUBJECT TO DISRUPTIONS

Alejandra Duenas, Dobrila Petrovic

*Control Theory and Applications Centre (CTAC), School of Mathematical and Information Sciences, Coventry University, Priory Street, Coventry, CV1 5FB. {A.Duenas, D.Petrovic}@coventry.ac.uk*

**Abstract:** In this paper, a new predictive-reactive approach to a parallel machine scheduling problem in the presence of uncertain disruptions is presented. The approach developed is based on generating a predictive parallel machine scheduling in such a way as to absorb the effects of possible uncertain disruptions through adding idle times to the job processing times. The uncertain disruption considered is material shortage and it is described using two parameters: number of disruption occurrences and disruption repair period. These parameters are specified imprecisely based on managerial subjective judgement and they are modelled and combined using fuzzy sets. If the impact of a disruption is too high to be absorbed by the predictive schedule, a rescheduling action is needed. In this new approach two rescheduling methods are applied, namely Left shift rescheduling and Building new schedules. The approach developed has been applied to solving a real-life scheduling problem of a pottery company. The results obtained have been satisfactory and showed the flexibility of the predictive-reactive approach.

**Key words:** production scheduling, rescheduling, parallel machines, uncertainty, fuzzy sets, dispatching rules

## 1. INTRODUCTION

Production scheduling typically defined as optimal or near optimal allocation of scarce resources, usually machines, to tasks over time [1], has been a topic that has attracted a wide interest of both academics and practitioners in the last fifty years. Complexity of production scheduling problems is caused by variety of machine configurations (e.g., single machine, parallel machines, flow shops, job shops), large scale dimensions (including number of machines and number of jobs to be scheduled), a wide range of parameters involved (job release dates, processing times, due dates, machine setup times, priorities of jobs, etc.), and uncertainty inherent in some parameters.

In addition, in most real life production environments, scheduling is an on going process where various disruptions in both external business and internal production conditions may occur dynamically and cause deviations from the initially generated schedule. Most often, these disruptions are uncertain. The importance of considering these uncertainties and developing rescheduling methods as a response to uncertain disruptions has been recognised mainly in the last decade. Viera et al. [2] reviewed rescheduling strategies, policies and methods. Aytug et al. [3] defined a taxonomy of production uncertainties based on four dimensions: 1) cause of uncertainty, such as material, process, resource, tooling and personnel, 2) context, i.e., environmental situation, 3) impact, e.g. on processing time, starting and completion times, material availability, quality, etc., and 4) inclusion in scheduling (predictive or reactive). The authors classified different approaches to scheduling in the presence of uncertainty into three groups: reactive scheduling, robust scheduling and predictive-reactive scheduling.

O'Donovan et al.[4] presented a single machine scheduling method that adds on idle time to the completion times of the scheduled jobs in order to absorb the impact of machine breakdowns. The machine

breakdowns are treated as stochastic processes modelled by two parameters, namely mean time between failures and mean time for repair. The objective was to maximise the predictability of the realised schedule by estimating the machine failures effects on the schedule and increasing the estimated job completion times correspondingly. Mehta and Uzsoy [5] developed an approach to predictable scheduling of a single machine subject to random machine breakdowns with the objective to absorb disruptions without affecting planned activities. They defined different measures of schedule predictability that are based on comparisons of the predictive schedule and the realised schedule completion times. In [6] four sources of production disturbances were identified and investigated, including: 1) incorrect work, 2) machine breakdowns, 3) rework due to a quality problem and 4) rush orders. They were assumed to be deterministic in the sense that their impact on the machine work time was determined precisely. A new rescheduling method for a job shop under random disruptions was proposed in [7]. Two measures of the rescheduling performance were considered simultaneously, including efficiency, measured by makespan, and stability, measured as the deviation from the initial schedule. In [8], a genetic algorithm based rescheduling method that considered both efficiency and stability criteria was developed and analysed. A conclusion was made that generating more stable schedules did not have a severe impact on scheduling efficiency.

In this paper, a new predictive-reactive approach to a parallel machine scheduling in the presence of uncertain disruptions is presented. A parallel machine scheduling problem is common in practice and considers a number of jobs to be processed on parallel machines. The objective is to determine the jobs allocation to the machines and the sequence of the jobs on each machine in order to optimise certain criteria [1].

The developed approach is based on generating a predictive parallel machine schedule using dispatching rules. The predictive schedule is designed to absorb the effects of a possible disruption through adding idle time to the job processing times. The idle time added is equal to the approximated repair time needed to recover from a disruption during the processing of a certain job.

Usually, the idle time to be added to the job processing times is determined assuming that the disruptions are random, and is represented by a probability distribution. The probability distributions are derived based on historical data. This requires a valid hypothesis that the data collected are complete and unbiased. However, very often in practice there is no evidence on the events recorded or there is lack of evidence, or lack of confidence in the evidence, or the evidence might have been recorded inconsistently, and therefore the use of concepts of probability theory might not be appropriate. In these situations, uncertain disruptions may be specified based on experience and managerial subjective judgement. It may be convenient to use natural language expressions in the specifications. It has been shown in a large body of literature that fuzzy sets theory provides a suitable framework for representing uncertainties in decision making problems where intuition and subjective judgements play an important role [9, 10]. In this paper, a new approach to dealing with uncertain disruptions is proposed where the disruption is specified imprecisely and modelled by fuzzy sets.

However, if the impact of a disruption is too high to be absorbed by the predictive schedule, a rescheduling action is needed. Reactive scheduling or rescheduling is defined as the process of modifying a schedule when a disruption occurs while executing the schedule in the shop floor. Rescheduling can involve either modification of the current schedule or generation of a new schedule. The developed predictive-reactive scheduling approach is applied to a real-life scheduling problem identified in collaboration with a manufacturing pottery company.

The paper is organised as follows. The problem of scheduling of parallel machines in the presences of uncertain disruptions is described in Section 2, while the new approach to predictive-reactive scheduling is presented in Section 3. The real-life scheduling problem is presented in Section 4, while the analysis of the results obtained is given in Section 5. Finally, conclusions and directions for future work are outlined in Section 6.

## 2. PROBLEM STATEMENT

A typical problem of identical parallel machines scheduling is stated as follows [1]:  $N$  jobs,  $J_j, j = 1, \dots, N$ , have to be scheduled on  $M$  machines  $M_i, i = 1, \dots, M$ , where each job is independent and nonpreemptive, and the machines are identical, i.e., they have the same speed. Hence, the jobs can be processed on any machine and their processing times are  $p_j, j = 1, \dots, N$ . The objective is to find the job's allocation for each machine that minimises the makespan, i.e. the time that takes to complete all the jobs. The makespan is defined as follows:

$$C_{\max} = \max\{C_j \mid j = 1, \dots, N\} \quad (1)$$

where  $C_j$  is the completion time of job  $J_j$ .

In this paper, the typical scheduling problem defined above is generalised in the following way. Some of the jobs to be scheduled can be processed only on predetermined machines; formally stated, job  $J_j, j = 1, \dots, N$ , can be processed on subset  $M_j$  of the  $M$  machines only.

In addition, it is assumed that the schedule will be realised in the presence of uncertain disruptions caused by material shortages. These disruptions have an adverse effect on the job processing times. In the case when the disruption is far too long or it cannot be repaired at all, the jobs affected are removed from the schedule. The remaining jobs are rescheduled. The reactive scheduling problem is considered as a multi-criteria optimisation problem. The two objectives to optimise are as follows: minimisation of the makespan  $C_{\max}$  (efficiency objective), and minimisation of the instability  $IST$  of the reactive schedule  $RS$ .

## 3. PREDICTIVE-REACTIVE SCHEDULING OF IDENTICAL PARALLEL MACHINES

A parallel machine schedule is generated following two steps. In the first step, a predictive schedule is determined, where idle times are added to jobs' processing times and a combination of dispatching rules namely the Least Flexible Job First (LFJ) and the Longest Processing Time (LPT) is applied. In the second step, two rescheduling methods, which modify the predictive schedule, are defined including Left-shift rescheduling and Building new schedules.

### 3.1 Predictive scheduling

Predictive schedules are generated in order to absorb anticipated disruptions. Traditionally, uncertain disruptions have been modelled using probability distributions that concern the occurrence of well defined events. Corresponding probabilities are assessed or estimated taking into consideration repetition of the events. For example, an uncertain disruption that has most often been treated in the literature is machine breakdowns, typically described by two parameters, namely mean time between failures and mean time for repair. These two parameters may be determined based on maintenance records. However, historic data may not be available for other sources of disruption. A typical example is the disruption caused by shortage of raw material. In this case, practitioners may estimate the disruption occurrences based on vague or imprecise knowledge, or accumulated experience. The corresponding data can be expressed using linguistic terms, such as 'the number of disruption occurrences is *much higher* than  $noc_1$  times per unit time period' or 'the material is usually delivered in *about*  $rp$  unit time periods' etc.

Such linguistic terms can be represented using fuzzy sets (see Appendix). For example, the above mentioned linguistic term '*much higher* than certain value' can be modelled by a discrete fuzzy set (Figure 1(a)), where each membership degree represents subjective belief or possibility that the disruption will occur certain number of times. On the other hand, the linguistic term '*about* certain value' can be represented by a

continuous fuzzy set (Figure 1(b)), where the disruption time period  $[b, c]$  has the highest possibility,  $a$  and  $d$  represent the lowest and the upper bound of possible disruption durations.

### 3.1.1 Calculation of idle times

It will be shown in this section, that uncertain data represented by fuzzy sets, such as number of material shortage occurrences and shortage duration, can be effectively combined. It is worth noting that probability theory is very restrictive on combining stochastic variables and corresponding probability distributions.

In the new approach developed, the job processing times are extended by inserting idle times. Each job  $J_j$  required one type of raw material and therefore might be affected by a shortage of that raw material only. Adding an idle time enables the uncertain disruptions to be absorbed in the schedule. The extended processing time  $pd_j$  for job  $J_j, j = 1, \dots, N$  that includes the idle time  $id_j$  is defined as follows:

$$pd_j = p_j + id_j, \quad j = 1, \dots, N \quad (2)$$

where  $id_j$  is the total disruption's repair time during the processing time of job  $J_j$ . The total disruption's repair time  $id_j$  is calculated as the product of the fuzzy number of disruption occurrences per unit time period and the fuzzy repair duration.

Having  $G$  different raw materials,  $m_g, g = 1, \dots, G$ , as inputs to the production process and a set  $NOC_g$  that contains the possible numbers of shortage occurrences of material  $m_g$  that is discrete and finite,  $NOC_g = \{noc_{g_1}, noc_{g_2}, \dots, noc_{g_K}\}$ , fuzzy set  $O_g$ , that represents the possible disruption occurrences per unit time is defined as:

$$O_g = \sum_{k=1}^K \mu_{O_g}(noc_{g_k}) / noc_{g_k} \quad (3)$$

where  $\mu_{O_g}(noc_{g_k}), k = 1, \dots, K$  is a possibility, subjectively determined, that there are  $noc_{g_k}$  disruption occurrences per unit time period, related to raw material  $m_g$ .

Fuzzy set  $R_g$ , that represents the repair duration i.e., delivery time of material  $m_g$ , has a continuous trapezoidal membership function  $\mu_{R_g}(tr_g)$  defined as follows:

$$R_g = \int_{tr_g} \mu_{R_g}(tr_g) / tr_g, \quad tr_g \in R^+$$

$$\mu_{R_g}(tr_g) = \begin{cases} 0 & \text{if } tr_g \leq a_g \\ \frac{tr_g - a_g}{b_g - a_g} & \text{if } a_g < tr_g \leq b_g \\ 1 & \text{if } b_g < tr_g \leq c_g \\ \frac{d_g - tr_g}{d_g - c_g} & \text{if } c_g < tr_g \leq d_g \\ 0 & \text{if } tr_g > d_g \end{cases} \quad (4)$$

In order to calculate the total disruption repair time per unit time period, a product of discrete fuzzy set  $O_g$  and continuous fuzzy set  $R_g$  is determined using an approach suggested in [11]. The product is calculated as a level 2 fuzzy set  $O_g \otimes R_g$ , i.e., a fuzzy set whose elements are standard fuzzy sets. The elements of  $O_g \otimes R_g$  are fuzzy sets  $noc_{g_k} \tilde{\times} R_g, k = 1, \dots, K$  calculated as a product of scalar  $noc_{g_k}$  and fuzzy set  $R_g$ . The

product is determined using the Extension principle [12], one of the most important principles in fuzzy sets theory, which allows the generalisation of classical mathematical concepts in the fuzzy sets framework.

In this case the membership function  $\mu_{noc_{g_k} \tilde{\times} R_g}$  has also a trapezoidal form and is calculated as follows:

$$\mu_{noc_{g_k} \tilde{\times} R_g}(tr_g) = \begin{cases} 0 & \text{if } tr_g \leq a_g \cdot noc_{g_k} \\ \frac{tr_g - (a_g \cdot noc_{g_k})}{(b_g - a_g) \cdot noc_{g_k}} & \text{if } a_g \cdot noc_{g_k} < tr_g \leq b_g \cdot noc_{g_k} \\ 1 & \text{if } b_g \cdot noc_{g_k} < tr_g \leq c_g \cdot noc_{g_k} \\ \frac{(d_g \cdot noc_{g_k}) - tr_g}{(d_g - c_g) \cdot noc_{g_k}} & \text{if } c_g \cdot noc_{g_k} < tr_g \leq d_g \cdot noc_{g_k} \\ 0 & \text{if } tr_g > d_g \cdot noc_{g_k} \end{cases} \quad (5)$$

The possibility of the total disruption repair time per unit time period being  $noc_{g_k} \tilde{\times} R_g$  is  $\mu_{O_g}(noc_{g_k})$ .

Formally, the total disruption repair time per unit time period  $O_g \otimes R_g$ , can be represented as level 2 fuzzy set:

$$O_g \otimes R_g = \sum_{k=1}^K \mu_{O_g}(noc_{g_k}) / noc_{g_k} \tilde{\times} R_g$$

In order to determine a crisp idle time to add to the job processing times, it is necessary to transform this level 2 fuzzy set into a standard fuzzy set, and then to defuzzify it. The method used to transform a level 2 fuzzy set into an ordinary fuzzy set is the s-fuzzification proposed by Zadeh[12] as follows:

$$s - fuzzif(O_g \otimes R_g) = \int_{tr_g} \mu_{s-fuzzif(O_g \otimes R_g)}(tr_g) | tr_g, tr_g \in R^+$$

where

$$\mu_{s-fuzzif(O_g \otimes R_g)}(tr_g) = \sup_{k=1, \dots, K} \mu_{O_g}(noc_{g_k}) \cdot \mu_{noc_{g_k} \tilde{\times} R_g}(tr_g) \quad (6)$$

The next step is to defuzzify the standard fuzzy set  $s - fuzzif(O_g \otimes R_g)$ , i.e., to find a scalar that represents the fuzzy set most appropriately [9]. The defuzzification method applied is the centroid method that finds an element  $td_g$  in the support of the fuzzy set  $s - fuzzif(O_g \otimes R_g)$  at which a line perpendicular to the axis passes through the centre of the area formed by the corresponding membership function. Value  $td_g$ , which is used as a crisp representation of the disruption's repair duration per unit time period, is calculated using the following expression:

$$td_g = \frac{\int_{tr_g \in R^+} tr_g \cdot \mu_{s-fuzzif(O_g \otimes R_g)}(tr_g)}{\int_{tr_g \in R^+} \mu_{s-fuzzif(O_g \otimes R_g)}(tr_g)} \quad (7)$$

In order to calculate the idle time  $id_j$  to be added to the initial processing time  $p_j$  of job  $J_j$ , the total disruption's repair time per unit time period is multiplied by the job processing time as follows:

$$id_j = td_g \cdot p_j \quad (8)$$

### 3.1.2 Dispatching Rules

Once the idle times are added to the initial processing times the schedule of the parallel machines is generated. According to Pinedo[1], one of the most common methods used is dispatching or priority rules. These rules are heuristics that have low computational complexity and are easy to implement. The selection of the rules to be used depends on the objective function of the problem and the machine environments under consideration [13]. For this problem two dispatching rules, the Least Flexible Job First (LFJ) and the Longest Processing Time (LPT) are combined.

The LFJ rule allocates the job that can be processed on the smallest number of machines, to the machine that is freed. One disadvantage of this rule is that if there is a tie, i.e., more than one job can be allocated, it is broken arbitrarily. In order to avoid this, another rule that has proved to be effective for minimising the makespan, the LPT rule, is combined with the LFJ rule. The LPT rule assigns, whenever a machine is freed, the job with the longest processing time. In other words, if a tie occurs when the LFJ rule is applied the job that has the longest processing time is selected.

In this way, the predictive schedule, *PS*, that takes into account uncertain disruptions is generated.

## 3.2 Reactive scheduling

There are occasions when the predictive schedule is unable to absorb the impacts of disruptions. In other words, the disruption's repair time can be too long or the disruption cannot be recovered at all. Since, in this approach the disruption considered is material shortage, when a material shortage cannot be recovered the jobs that use that specific material have to be removed from the sequence. Therefore, it is necessary to reschedule. In the developed approach two rescheduling methods are applied and compared:

1. Left-shift rescheduling: the start times of the remaining jobs are left-shifted to the time when the high impact disruption occurs, keeping the same sequence of the jobs as in the predictive schedule *PS*. It is assumed that a high impact disruption affects one of the machines only, and the left-shift rescheduling is applied to that particular machine. Therefore, rescheduling in this case can be treated as a single machine scheduling.
2. Building new schedules: once a high impact disruption occurs, a new schedule is generated considering the jobs that have not been processed. This new schedule is built using the same scheduling algorithm proposed for generating an initial identical parallel machines schedule, but in this case with a smaller number of jobs.

A new criterion of stability/instability of the reactive schedule *RS* is considered, in addition to the initial criterion of schedule efficiency (makespan) used in generating the predictive schedule *PS*. The instability *IST* of the reactive schedule *RS* is measured as the completion time deviations between the predictive schedule *PS* and the reactive schedule *RS* as follows [14]:

$$IST(RS) = \sqrt{\sum_{j=1}^N (C_j(PS) - C_j(RS))^2} \quad (9)$$

In most of the rescheduling approaches proposed in the literature, rescheduling is applied on a periodic basis. However, in the approach presented in this paper rescheduling is only applied when it is assumed that the disruption effect cannot be absorbed by the predictive schedule.



## 4. A REAL-LIFE SCHEDULING PROBLEM

The predictive-reactive approach presented in the previous section is applied to a scheduling problem of the Denby Pottery Company Ltd. in UK. The company has been involved in the pottery industry for almost 200 years and manufactures a wide range of ceramic tableware products. One of the most important processes in the pottery industry is glazing. The glazing section is divided into three identical ‘flowlines’ that are considered as identical parallel machines. A job is defined as the number of items of a specific product to be produced. The number of jobs can vary according to the production plan.

At present, the glazing section leader (decision maker) builds the schedule by hand dealing with approximately 30 jobs on weekly basis. In the glazing section not all the jobs can be produced on any of the three machines. Therefore, the glazing scheduling problem is considered as an identical parallel machine scheduling problem with  $N = 30$  jobs and  $M = 3$  machines when job  $J_j$ ,  $j = 1, \dots, 30$ , can be processed only on subset  $M_j$  of the 3 machines. After observing the production process, it was concluded that one of the disruptions that had the greatest impact on the scheduling execution occurs when a machine runs out of a glaze. According to the glazing section leader, the only time when they know if they have run out of a certain glaze is once the job has been started.

Table 1 presents a sample of the data used consisting of 10 jobs only, where jobs  $J_1$ ,  $J_2$ ,  $J_5$ ,  $J_9$  and  $J_{10}$  can only be processed on machine  $M_1$  and  $M_3$ . The pattern refers to the glaze that is needed to process the job and the item refers to the shape of the product to be processed. The unit time is different for each product and refers to the processing time of 100 items. The processing time  $p_j$ ,  $j = 1, \dots, 10$  refers to the time required for processing job  $J_j$ , where the batch size of job  $J_j$  is not necessarily 100 items. For example, job  $J_1$  contains 250 items and, consequently, the processing time  $p_1 = 10 = (4 \cdot 250 / 100)$  hours. The data given in the table are hypothetical as the real data cannot be presented due to confidentiality.

As previously defined, in order to generate a predictive schedule, it is necessary to add on idle time  $id_j$  to the initial processing time  $p_j$  of job  $J_j$ ,  $j = 1, \dots, 30$ .

Historical data of numbers of glaze shortages and the time periods required to obtain the glazes are not available. However, the glazing section leader can specify imprecisely the number of shortage occurrences per unit time period for each glaze and the time that is usually needed to get the glaze from the glaze production department.

### 4.1 Predictive scheduling

In order to generate a predictive schedule, the total disruption’s repair time per unit time period is calculated for each glaze. To illustrate this procedure, the Regency green glaze, enumerated as  $g = 1$ , is considered where the possible occurrences of glaze shortage per unit time period are 1, 2, 3 or 4 with possibilities 0.7, 0.95, 0.4 and 0.25, respectively, i.e.,  $O_1 = \{0.7/1 + 0.95/2 + 0.4/3 + 0.25/4\}$ , represented in Figure 2.

Fuzzy set  $R_1$  represents the imprecise repair duration  $tr_1$  of material  $m_1$  (Regency green glaze), shown in Figure 3. Both fuzzy sets are determined based on the information obtained in the glaze production department at Denby Pottery.

Figure 4 shows the level 2 fuzzy set  $O_1 \otimes R_1$ , that represents the total disruption repair time per unit time period for glaze  $m_1 = 1$ , calculated as the product of the two fuzzy sets  $O_1$  and  $R_1$ . Four fuzzy values of the total disruption repair time per unit time period with associated possibilities are presented.

Once the total repair duration time per unit time period is calculated as a level 2 fuzzy set, it is transformed into a standard set using s-fuzzification and then defuzzified, leading to the crisp total repair duration time,  $td_1 = 3$ . In order to find the total disruption’s repair time of Regency green glaze shortage during the processing of a job  $J_1$  that uses that specific glaze,  $td_1$  is multiplied by the processing time  $p_1$  of

the job. The unit time for the Regency green glaze is 4 hours, and considering job  $J_1$  initial processing time  $p_1 = 10$ ,  $id_j$  is calculated as follows:

$$id_1 = \frac{3 \cdot 10}{4} = 7.5 \text{ hours}$$

Therefore,  $pd_1 = 10 + 7.5 = 17.5$  hours. This procedure is repeated for each glaze and jobs that might be affected by the shortage of the glaze under consideration. Once, all the processing times are extended by adding the corresponding idle times the schedule that minimises the makespan  $C_{\max}$  is generated.

## 4.2 Reactive scheduling

In practice, there will be occasions when the disruption cannot be absorbed; for example, if the glaze has low quality and it is impossible to apply it and the glaze production department informs the glazing section leader that the repair time i.e., the time to deliver the glaze is too long. In this case, rescheduling has to be applied. Two rescheduling techniques are proposed: Left-shift rescheduling and Building new schedules as defined in Section 3.

## 5. RESULTS ANALYSIS

Two analyses are carried out: (1) to evaluate the effects of the imprecisely specified disruptions on the jobs' sequence and completion times, (2) to investigate the effects of rescheduling.

### 5.1 Disruptions impact analysis

The aim of this analysis is to investigate the effects of changing fuzzy repair duration time on the jobs' sequence and completion times. As defined in Section 4, the total disruption' repair time per unit time period is determined by the possibility of disruption occurrences per unit time period and the repair duration. The repair duration is linguistically represented as '*about  $b$  to  $c$  unit time periods*' and modelled by a fuzzy set with a trapezoidal membership function specified by four parameters  $a$ ,  $b$ ,  $c$ , and  $d$ .

In order to analyse the effects of changing the disruption duration period, the parameters  $a$ ,  $b$ ,  $c$ , and  $d$  are varied in two ways: (1) uncertainty in disruption duration period is increased by increasing the distance between  $b$  and  $c$  and (2) disruption duration period is increased by shifting the domain to the right i.e., by increasing the values of parameters  $a$ ,  $b$ ,  $c$  and  $d$ .

#### 5.1.1 Effect of increasing uncertainty in disruption duration period

Tables 2, 3 and 4 show the jobs' sequence, completion times and  $C_{\max}$  obtained for three different sets of parameter values: (1)  $a = 1$ ,  $b = 2$ ,  $c = 3$  and  $d = 4$ , (2)  $a = 1$ ,  $b = 2$ ,  $c = 4$  and  $d = 5$ , and (3)  $a = 1$ ,  $b = 3$ ,  $c = 6$  and  $d = 8$ , respectively.

As expected, the results obtained showed that changing uncertainty in duration repair period has an adverse impact on the makespan. For example, if the distance between  $b$  and  $c$  is increased from 1 hour ( $b = 2$  and  $c = 3$ ) to 2 hours ( $b = 2$  and  $c = 4$ ) the makespan increases from 88.5 to 98.9 hours, this represents an increment of 10.4 hours (12%). If the distance between  $b$  and  $c$  is increased from 1 hour ( $b = 2$  and  $c = 3$ ) to 3 hours ( $b = 3$  and  $c = 6$ ) the makespan increases from 88.5 to 130 hours, which represents an increment of 41.5 hours (47%). Consequently, it can be concluded that widening the distance between  $b$  and  $c$  does not have a linear impact on the makespan deterioration.

It may be interesting to analyse the effects of widening the distance between  $b$  and  $c$  on the glaze disruption's repair duration per unit time period  $td_g$ ,  $g = 1, \dots, 8$ . It can be seen, in Table 5, that the glaze disruption's repair duration per unit time period  $td_g$ , directly depends on the uncertainty in the repair duration; the more uncertainty in the disruption repair duration, the longer the idle time to be added to the processing time.

### **5.1.2 The effects of increasing the disruptions duration period**

The jobs' sequence, completion times and  $C_{\max}$  are obtained for three different disruption periods, specified by: (1)  $a = 2$ ,  $b = 3$ ,  $c = 4$  and  $d = 5$ , (2)  $a = 3$ ,  $b = 4$ ,  $c = 5$  and  $d = 6$ , and (3)  $a = 4$ ,  $b = 5$ ,  $c = 6$  and  $d = 7$  and the results obtained are presented in Tables 6, 7 and 8, respectively.

If possible disruption duration periods are increased by 1 hour, i.e., the domain of the corresponding fuzzy set is 1 hour left-shifted, starting from  $a = 1$  to  $a = 2$ , the makespan  $C_{\max}$  increases from 88.5 (Table 2) to 98.9 hours (Table 6); this represents an increment of 10.4 hours (12%). Similarly, if the possible disruption periods are increased by 2 hours, i.e.,  $a, b, c, d$  is changed from  $a = 1$  to  $a = 3$ ,  $b = 2$  to  $b = 4$ ,  $c = 3$  to  $c = 5$  and  $d = 4$  to  $d = 6$  the makespan  $C_{\max}$  increases from 88.5 to 108.7 hours, that represents an increment of 20.2 hours (23%). If the increment is 3 hours, i.e.,  $a$  is changed from  $a = 1$  to  $a = 4$  and all other parameters correspondingly, the makespan  $C_{\max}$  increases from 88.5 to 119.1 hours, and this is an increment of 30.6 hours (35%). Comparing this results with those found when the distance between  $b$  and  $c$  was increased (Table 2 to Table 6), it appears that the impact of increasing the uncertainty in disruption duration is higher than the impact of increasing the possible disruptions durations.

It can also be seen that the jobs' sequence and completion times when  $a = 1$ ,  $b = 2$ ,  $c = 4$  and  $d = 5$  are the same as for  $a = 2$ ,  $b = 3$ ,  $c = 4$  and  $d = 5$ .

It may be of interest to analyse the effects of increasing the possible disruption duration periods on the glazes' disruption's repair duration per unit time period  $td_g$  (see Table 9).

If Table 5 and Table 9 are compared it is possible to see that the impact of increasing uncertainty in disruption duration period, i.e., increasing the distance between  $b$  and  $c$  is higher than the impact of increasing the possible disruption duration. In other words, increments in uncertainty in the disruption duration periods may cause more changes in the idle times to be added to the processing times than increments in the disruption duration.

## **5.2 Analysis of rescheduling impact**

Two rescheduling methods are proposed: Left-shift rescheduling and Building new schedules. The aim of this analysis is to determine which of these methods yields better results in terms of the two objectives considered, namely efficiency and stability. In order to analyse the impact of rescheduling it is necessary to assume a scenario where high impact disruptions occur.

In the scenario under consideration, a bad quality in the Greenwich glaze is detected by the section leader at the moment of producing the first job that involves this glaze. However, the glaze production department is not able to deliver the glaze in a feasible time, and therefore it is necessary to remove all the jobs that use this glaze.

Table 10 shows different glazes  $m_g$ ,  $g = 1, \dots, 8$  considered in this problem, as well as the values of the parameters  $a$ ,  $b$ ,  $c$ , and  $d$  used in order to generate the schedule. As described in Section 4, the disruption's repair time per unit time period  $td_g$  is calculated for each glaze. The jobs' sequence and completion times are obtained as presented in Table 11.

The jobs that use Greenwich glaze are  $J_2$ ,  $J_{12}$  and  $J_{22}$ . In terms of starting times,  $J_2$  starts processing at 93.7 hours,  $J_{12}$  starts processing at 91.2 hours and  $J_{22}$  starts processing at 89.5 hours. The jobs that use Greenwich

glaze will be cut off from the schedule and the first job to be removed is job  $J_{22}$ . When Left-shifting rescheduling is applied, after the jobs are cut off, the machines where they are expected to be processed ( $M_1$  and  $M_3$ ) are treated as single machines. The schedule obtained after applying the Left-shifting rescheduling method is presented in Table 12.

As it can be seen, in this case the makespan was not affected since it was determined by machine  $M_2$  and this machine was not affected by the disruption. In this case, the objective related to efficiency has not been affected.

The next objective to measure is the instability defined by formula (9):

$$IST(RS) = \sqrt{\sum_{j=1}^{30} (C_j(PS) - C_j(RS))^2} = 166.99$$

Since jobs  $J_2$ ,  $J_{12}$  and  $J_{22}$  are cancelled in the reactive schedule  $RS$ , in order to calculate  $IST(RS)$  the completion times are considered to be equal to zero i.e.,  $C_2(RS) = 0$ ,  $C_{12}(RS) = 0$  and  $C_{22}(RS) = 0$ .

When the Building new schedules method is applied, the new schedule is generated with all the jobs that have not been processed yet, considering the moment of disruption to be the jobs starting time. In the predictive schedule  $PS$  (Table 11), the first job affected by the disruption is job  $J_{22}$  which starts processing at 89.5 hours, on machine  $M_3$ . Jobs  $J_5$  and  $J_{17}$  have started being processed and have a completion time of 91.2 and 108.2 hours on machines  $M_1$  and  $M_2$ , respectively. Therefore, the starting time for the new schedule to be built is 91.2 hours for machine  $M_1$ , 108.2 hours for machine  $M_2$  and 89.5 for machine  $M_3$ . Table 13 shows the schedule obtained after applying the building new schedules method.

As it can be seen, in this case the makespan,  $C_{\max}=139.6$ , is smaller than the one found by the Left-shifting rescheduling method,  $C_{\max}=144.9$ . This occurs because the Building new schedule method applies the rules LFJ and LPT which minimise the makespan while the Left-shifting rescheduling method does not consider makespan at all. Additionally, in order to determine which method generates a more stable schedule, the instability of the new schedule is calculated using formula (9) as follows:

$$IST(RS) = \sqrt{\sum_{j=1}^{30} (C_j(PS) - C_j(RS))^2} = 167.022$$

It can be seen that for the efficiency objective (makespan) the rescheduling method that performs better is the building new schedules method, while the Left-shifting method might perform better for the stability/instability objective. It may be concluded that selection of the rescheduling method to be applied depends on which of the objectives considered is more important, the efficiency or stability/instability.

Another issue worth considering is the addition of new jobs, as it is often the case in practice that, when a disruption occurs, the production planning department decides to produce some new jobs since some initially planned jobs cannot be produced. In this scenario, for illustrative purposes, it is decided to consider two new jobs with the data given in Table 14:

In the Left-shifting rescheduling method, the new jobs can only be added after all the jobs scheduled initially have been completed. Table 15 shows one possible reactive schedule  $RS$  with two new jobs  $J_{31}$  and  $J_{32}$ , generated using dispatching rules LFJ and LPT.

The instability is calculated to be  $IST(RS) = 166.99$ . Since jobs  $J_{31}$  and  $J_{32}$  do not exist in the predictive schedule  $PS$ , in order to calculate  $IST(RS)$  the completion times are considered to be equal to zero i.e.,  $C_{31}(PS) = 0$  and  $C_{32}(PS) = 0$ .

When the Building new schedules method is applied, the new schedule is generated for all the jobs that have not been processed yet, including the two new jobs  $J_{31}$  and  $J_{32}$ , and considering, as the jobs starting time, the moment of disruption. Table 16 shows the schedule obtained after applying the building new schedules method including the two new jobs. Makespan of the schedule  $C_{\max}$  is 153.1 while instability is  $IST(RS) = 168.89$ .

It can be seen that in this scenario when the two jobs are added, the rescheduling method that yields better results for both objectives, efficiency and instability, is the Left-shifting rescheduling.

Hence, the analyses carried out show the importance of selecting the appropriate rescheduling method taking into consideration both the decision maker requirements and preferences, and the objectives to be optimised.

## 6. CONCLUSION

A predictive-reactive approach to an identical parallel machine scheduling problem is presented, where a number of jobs has to be processed on parallel machines. The machines are considered to be identical, in other words, the machines have the same speed. The aim of the scheduling is to determine the jobs allocation to the machines and the sequence of the jobs on each machine in the presence of uncertain disruptions in the production environment in such a way as to minimise the makespan.

The new approach developed is based on generating a predictive parallel machine schedule using dispatching rules, where the predictive schedule is designed to absorb the effects of a possible uncertain disruption through adding idle times to the jobs' processing times.

Uncertain disruption is specified imprecisely using two parameters, namely number of disruptions occurrences and disruption repair period. They are modelled using fuzzy sets and combined effectively into level 2 fuzzy sets. If the impact of a disruption is too high to be absorbed by the predictive schedule a rescheduling action is needed. The predictive-reactive scheduling approach is applied to a real life-scheduling problem identified in collaboration with a manufacturing pottery company. The use of fuzzy sets in modelling material shortage disruptions proved to be beneficial in this case when there are no historical data and subjective managerial judgement can be used. Two rescheduling methods namely Left-shifting and building new schedules have proven to be of benefit when high impact disruptions occur.

The results obtained show that uncertainty in disruptions caused by shortage of a glaze or low quality of the glaze may have a higher impact on the schedule's execution than repair time, i.e., the time needed for the glaze production department to deliver the glaze. Importance of selecting the appropriate rescheduling method, according to the decision maker requirements and preferences, and the objectives to be optimised is demonstrated. The results obtained using the rescheduling methods are satisfactory and showed the model's flexibility since the rescheduling methods can also be used when new jobs arrive.

Further work will be undertaken including:

- Investigation of different techniques to measure level of uncertainty, represented by fuzzy sets in order to quantify the impact of uncertain disruptions on the schedule performance,
- Development of a decision-making method for determining whether to perform rescheduling at a particular disruption and which rescheduling method to use,
- Development of a simulation model of realised schedules in order to measure the level of predictability of the predictive schedule, i.e., the quality of the schedule in terms of absorbing the effects of a disruption.
- Analysis of possible improvements in scheduling that might be gained at the pottery company by using the method proposed.

## ACKNOWLEDGEMENTS

This research is supported by Engineering and Physical Sciences Research Council (EPSRC), grant no. GR/R95326/01. This support is gratefully acknowledged. We also acknowledge the support of the industrial collaborator the Denby Pottery Company Ltd.

## APPENDIX

Definition of a fuzzy set [12]. Let  $X$  denotes a universal set with elements denoted as  $x$ . A fuzzy set  $A$  in  $X$  is characterised by a membership function  $\mu_A(x)$ , where  $\mu_A(x):X \rightarrow [0,1]$  associates each element  $x$  with a degree of membership of  $x$  in  $A$ . In the case when  $x$  is discrete and finite  $x = \{x_1, x_2, \dots, x_k\}$ , fuzzy set  $A$  may be denoted by  $A = \sum_{k=1}^K \mu_A(x_k) / x_k$ . When  $X = \{x\}$  is continuous, fuzzy set  $A$  may be represented by  $A = \int_x \mu_A(x) / x$ .

## REFERENCES

1. Pinedo M., *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, 2002.
2. Vieira G.E., Herrmann J.W., Lin E., Rescheduling manufacturing systems: a framework of strategies, policies, and methods, *Journal of Scheduling*, Vol. 6, pp. 39-62, 2003.
3. Aytug H., Lawley M.A., McKay K., Mohan S., Uzsoy R., Executing production schedules in the face of uncertainties: A review and some future directions, *European Journal of Operational Research*, Vol. 161, pp.86-110, 2005.
4. O'Donovan R., Uzsoy R., McKay K.N., Predictable scheduling of a single machine with breakdown and sensitive jobs, *International Journal of Production Research*, Vol. 37, No. 18, pp. 4217-4233, 1999.
5. Mehta S.V., Uzsoy R., Predictable scheduling of a single machine subject to breakdowns, *International Journal of Computer Integrated Manufacturing*, Vol. 12, No. 1, 15-38, 1999.
6. Li H., Li Z., Li L.X. Hu B., A production rescheduling expert simulation system, *European Journal of Operational Research*, Vol. 124, pp. 283-293, 2000.
7. Abumaizar R.J., Svestka J.A., Rescheduling job shops under random disruptions, *International Journal of Production Research*, Vol. 35, No. 7, pp. 2065-2082, 1997.
8. Rangaritratsamee, R., Ferrell W.G., Kurz M.B., Dynamic rescheduling that simultaneously considers efficiency and stability, *Computers and Industrial Engineering*, Vol. 46, pp. 1-15, 2004.
9. Ruspini E.H., P.P. Bonissone and W. Pedrycz, *Handbook of Fuzzy Computation*, Institute of physics publishing, USA, 1998.
10. Zimmermann H-J., *Fuzzy Set Theory-and Its Applications*, Third Edition, Kluwer Academic Publishers, 1996.
11. Petrovic D., Petrovic R., Vujosevic M., Fuzzy models for the newsboy problem, *International Journal of Production Economics*, Vol. 45, pp. 435-441, 1996.
12. Zadeh L.A., Fuzzy Sets, *Information and Control*, Vol. 8, pp. 338-353, 1965.
13. Panwalkar S.S, Iskander W., A survey of scheduling rules, *Operations Research*, Vol. 25, No. 1, pp. 45-61, 1977.
14. Papoulis A., *Probability, random variables and stochastic processes*, Third Edition, McGraw-Hill, Inc., 1991.

# An approach to predictive-reactive scheduling of parallel machines subject to disruptions

Table 1. Identical parallel machine scheduling data for 10 jobs

| job $J_j$ | pattern       | item              | unit time<br>(hours)/ 100<br>items | processing<br>time $p_j$ (hours) | $M_1$ | $M_2$ | $M_3$ |
|-----------|---------------|-------------------|------------------------------------|----------------------------------|-------|-------|-------|
| 1         | Regency green | Large Jug         | 4                                  | 10                               | ✓     | ×     | ✓     |
| 2         | Greenwich     | Large Jug         | 4                                  | 2                                | ✓     | ×     | ✓     |
| 3         | Marrakesh     | Sauce Boat        | 6                                  | 5                                | ✓     | ✓     | ✓     |
| 4         | Regency green | Teapot Base       | 4                                  | 7                                | ✓     | ✓     | ✓     |
| 5         | Fire          | Small Jug         | 5                                  | 2                                | ✓     | ×     | ✓     |
| 6         | Energy        | Teapot (Classic)  | 3                                  | 3                                | ✓     | ✓     | ✓     |
| 7         | Harlequin     | Small Teapot Base | 4                                  | 8                                | ✓     | ✓     | ✓     |
| 8         | Spirit        | Large Teapot Base | 6                                  | 4                                | ✓     | ✓     | ✓     |
| 9         | Fire          | Large Jug         | 5                                  | 1                                | ✓     | ×     | ✓     |
| 10        | Blue Jetty    | Sauce Jug         | 4                                  | 6                                | ✓     | ×     | ✓     |

Table 2. Jobs' sequence, completion times and  $C_{\max}$  when  $a = 1$ ,  $b = 2$ ,  $c = 3$  and  $d = 4$

| machine      | jobs' sequence   |      |      |      |      |      |      |      |      |      |      |      |
|--------------|------------------|------|------|------|------|------|------|------|------|------|------|------|
| $M_1$        | 20               | 21   | 11   | 22   | 12   | 25   | 5    | 9    | 4    | 23   | 13   | 6    |
| $M_2$        | 27               | 17   | 7    | 24   | 28   | 18   | 3    |      |      |      |      |      |
| $M_3$        | 30               | 10   | 1    | 2    | 15   | 29   | 19   | 14   | 8    | 26   | 16   |      |
| machine      | completion times |      |      |      |      |      |      |      |      |      |      |      |
| $M_1$        | 18.0             | 31.5 | 45.0 | 48.0 | 51.0 | 54.0 | 57.0 | 58.5 | 69.0 | 76.5 | 84.0 | 88.5 |
| $M_2$        | 16.0             | 32.0 | 48.0 | 58.5 | 68.5 | 78.5 | 86.0 |      |      |      |      |      |
| $M_3$        | 18.0             | 36.0 | 49.5 | 52.5 | 55.5 | 57.0 | 58.5 | 69.0 | 79.0 | 83.5 | 88.0 |      |
| $C_{\max} =$ | 88.5             |      |      |      |      |      |      |      |      |      |      |      |

Table 3. Jobs' sequence, completion times and  $C_{\max}$  when  $a = 1$ ,  $b = 2$ ,  $c = 4$  and  $d = 5$

| machine      | jobs' sequence   |      |      |      |      |      |      |      |      |      |      |  |
|--------------|------------------|------|------|------|------|------|------|------|------|------|------|--|
| $M_1$        | 20               | 21   | 11   | 25   | 15   | 22   | 2    | 19   | 18   | 14   | 3    |  |
| $M_2$        | 27               | 17   | 7    | 28   | 8    | 4    | 26   | 16   |      |      |      |  |
| $M_3$        | 30               | 10   | 1    | 5    | 12   | 29   | 9    | 24   | 23   | 13   | 6    |  |
| machine      | completion times |      |      |      |      |      |      |      |      |      |      |  |
| $M_1$        | 21.0             | 35.6 | 50.2 | 53.5 | 56.7 | 60.0 | 63.2 | 64.9 | 76.4 | 87.7 | 95.9 |  |
| $M_2$        | 18.0             | 36.0 | 54.0 | 65.5 | 77.0 | 88.4 | 93.2 | 98.1 |      |      |      |  |
| $M_3$        | 21.0             | 42.0 | 56.6 | 59.9 | 63.1 | 64.7 | 66.4 | 77.7 | 85.9 | 94.0 | 98.9 |  |
| $C_{\max} =$ | 98.9             |      |      |      |      |      |      |      |      |      |      |  |

Table 4. Jobs' sequence, completion times and  $C_{\max}$  when  $a = 1$ ,  $b = 3$ ,  $c = 6$  and  $d = 8$

| machine      | jobs' sequence   |      |      |      |       |       |       |       |       |       |       |       |
|--------------|------------------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|
| $M_1$        | 20               | 21   | 11   | 25   | 22    | 15    | 12    | 2     | 9     | 24    | 14    | 3     |
| $M_2$        | 27               | 17   | 7    | 28   | 8     | 23    | 13    | 6     |       |       |       |       |
| $M_3$        | 30               | 10   | 1    | 5    | 29    | 19    | 18    | 4     | 26    | 16    |       |       |
| machine      | completion times |      |      |      |       |       |       |       |       |       |       |       |
| $M_1$        | 30.0             | 48.0 | 66.0 | 70.0 | 74.0  | 78.0  | 82.0  | 86.0  | 88.0  | 102.0 | 116.0 | 126.0 |
| $M_2$        | 24.0             | 48.0 | 72.0 | 88.0 | 104.0 | 114.0 | 124.0 | 130.0 |       |       |       |       |
| $M_3$        | 30.0             | 60.0 | 78.0 | 82.0 | 84.0  | 86.0  | 102.0 | 116.0 | 122.0 | 128.0 |       |       |
| $C_{\max} =$ | 130              |      |      |      |       |       |       |       |       |       |       |       |

Table 5. Glaze disruption's repair duration per unit time

| Glaze $m_g$ | pattern       | $td_g$<br>(hours)                      |                                        |                                        |
|-------------|---------------|----------------------------------------|----------------------------------------|----------------------------------------|
| $g$         |               | $a = 1, b = 2,$<br>$c = 3$ and $d = 4$ | $a = 1, b = 2,$<br>$c = 4$ and $d = 5$ | $a = 1, b = 3,$<br>$c = 6$ and $d = 8$ |
| 1           | Regency green | 2                                      | 2.5                                    | 4                                      |
| 2           | Greenwich     | 2                                      | 2.5                                    | 4                                      |
| 3           | Marrakesh     | 2                                      | 2.5                                    | 4                                      |
| 4           | Fire          | 2                                      | 2.5                                    | 4                                      |
| 5           | Energy        | 2                                      | 2.5                                    | 4                                      |
| 6           | Harlequin     | 4                                      | 5                                      | 8                                      |
| 7           | Spirit        | 6                                      | 7.5                                    | 12                                     |
| 8           | Blue Jetty    | 8                                      | 10                                     | 16                                     |

Table 6. Jobs' sequence, completion times and  $C_{\max}$  when  $a = 2, b = 3, c = 4$  and  $d = 5$ 

| machine      | jobs' sequence   |      |      |      |      |      |      |      |      |      |      |
|--------------|------------------|------|------|------|------|------|------|------|------|------|------|
| $M_1$        | 20               | 21   | 11   | 25   | 15   | 22   | 2    | 19   | 18   | 14   | 3    |
| $M_2$        | 27               | 17   | 7    | 28   | 8    | 4    | 26   | 16   |      |      |      |
| $M_3$        | 30               | 10   | 1    | 5    | 12   | 29   | 9    | 24   | 23   | 13   | 6    |
| machine      | completion times |      |      |      |      |      |      |      |      |      |      |
| $M_1$        | 21.0             | 35.6 | 50.2 | 53.5 | 56.7 | 60.0 | 63.2 | 64.9 | 76.4 | 87.7 | 95.9 |
| $M_2$        | 18.0             | 36.0 | 54.0 | 65.5 | 77.0 | 88.4 | 93.2 | 98.1 |      |      |      |
| $M_3$        | 21.0             | 42.0 | 56.6 | 59.9 | 63.1 | 64.7 | 66.4 | 77.7 | 85.9 | 94.0 | 98.9 |
| $C_{\max} =$ | 98.9             |      |      |      |      |      |      |      |      |      |      |

Table 7. Jobs' sequence, completion times and  $C_{\max}$  when  $a = 3, b = 4, c = 5$  and  $d = 6$ 

| machine            |       | jobs' sequence   |      |      |      |      |       |       |       |       |      |       |
|--------------------|-------|------------------|------|------|------|------|-------|-------|-------|-------|------|-------|
| M <sub>1</sub>     | 20    | 21               | 11   | 25   | 15   | 5    | 12    | 29    | 9     | 24    | 14   | 3     |
| M <sub>2</sub>     | 27    | 17               | 7    | 28   | 8    | 23   | 13    | 6     |       |       |      |       |
| M <sub>3</sub>     | 30    | 10               | 1    | 22   | 2    | 19   | 18    | 4     | 26    | 16    |      |       |
| machine            |       | completion times |      |      |      |      |       |       |       |       |      |       |
| M <sub>1</sub>     | 24.0  | 39.7             | 55.5 | 59.0 | 62.5 | 66.0 | 69.5  | 71.2  | 73.0  | 85.2  | 97.5 | 106.2 |
| M <sub>2</sub>     | 20.0  | 40.0             | 60.0 | 73.0 | 86.0 | 94.7 | 103.5 | 108.7 |       |       |      |       |
| M <sub>3</sub>     | 24.0  | 48.0             | 63.7 | 67.2 | 70.7 | 72.5 | 85.5  | 97.7  | 103.0 | 108.2 |      |       |
| C <sub>max</sub> = | 108.7 |                  |      |      |      |      |       |       |       |       |      |       |

Table 8. Jobs' sequence, completion times and  $C_{\max}$  when  $a = 4, b = 5, c = 6$  and  $d = 7$ 

| Machine            | jobs' sequence   |      |      |      |      |       |       |       |       |       |       |       |       |
|--------------------|------------------|------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|
| M <sub>1</sub>     | 20               | 21   | 11   | 25   | 15   | 5     | 12    | 29    | 19    | 18    | 4     | 26    | 16    |
| M <sub>2</sub>     | 27               | 17   | 7    | 28   | 24   | 14    | 3     |       |       |       |       |       |       |
| M <sub>3</sub>     | 30               | 10   | 1    | 22   | 2    | 9     | 8     | 23    | 13    | 6     |       |       |       |
| machine            | completion times |      |      |      |      |       |       |       |       |       |       |       |       |
| M <sub>1</sub>     | 27.0             | 43.9 | 60.8 | 64.5 | 68.3 | 72.0  | 75.8  | 77.6  | 79.5  | 94.0  | 107.1 | 112.7 | 118.4 |
| M <sub>2</sub>     | 22.0             | 44.0 | 66.0 | 80.5 | 93.6 | 106.7 | 116.1 |       |       |       |       |       |       |
| M <sub>3</sub>     | 27.0             | 54.0 | 70.9 | 74.6 | 78.4 | 80.3  | 94.8  | 104.1 | 113.5 | 119.1 |       |       |       |
| C <sub>max</sub> = | 119.1            |      |      |      |      |       |       |       |       |       |       |       |       |



# An approach to predictive-reactive scheduling of parallel machines subject to disruptions

Table 9. Glaze disruption's repair duration per unit time

| glaze $m_g$ | pattern       | $td_g$<br>(hours)                      |                                        |                                        |
|-------------|---------------|----------------------------------------|----------------------------------------|----------------------------------------|
|             |               | $a = 2, b = 3,$<br>$c = 4$ and $d = 5$ | $a = 3, b = 4,$<br>$c = 5$ and $d = 6$ | $a = 4, b = 5,$<br>$c = 6$ and $d = 7$ |
| 1           | Regency green | 2.5                                    | 3                                      | 3.5                                    |
| 2           | Greenwich     | 2.5                                    | 3                                      | 3.5                                    |
| 3           | Marrakesh     | 2.5                                    | 3                                      | 3.5                                    |
| 4           | Fire          | 2.5                                    | 3                                      | 3.5                                    |
| 5           | Energy        | 2.5                                    | 3                                      | 3.5                                    |
| 6           | Harlequin     | 5                                      | 6                                      | 7                                      |
| 7           | Spirit        | 7.5                                    | 9                                      | 10.5                                   |
| 8           | Blue Jetty    | 10                                     | 12                                     | 14                                     |

Table 10. Glazes considered

| glaze $m_g$ | pattern       | $a$ | $b$ | $c$ | $d$ | $td_g$<br>(hours) |
|-------------|---------------|-----|-----|-----|-----|-------------------|
| $g$         |               |     |     |     |     |                   |
| 1           | Regency green | 2   | 5   | 7   | 10  | 9                 |
| 2           | Greenwich     | 1   | 2   | 3   | 4   | 4.5               |
| 3           | Marrakesh     | 5   | 8   | 13  | 16  | 6.5               |
| 4           | Fire          | 4   | 8   | 14  | 18  | 10                |
| 5           | Energy        | 1   | 3   | 6   | 9   | 3                 |
| 6           | Harlequin     | 3   | 6   | 11  | 13  | 6                 |
| 7           | Spirit        | 8   | 12  | 16  | 20  | 9                 |
| 8           | Blue Jetty    | 1   | 2   | 5   | 6   | 12                |

Table 11. Predictive schedule PS

| machine                         |       |      |      |      | jobs' sequence   |       |       |       |       |       |       |
|---------------------------------|-------|------|------|------|------------------|-------|-------|-------|-------|-------|-------|
| M <sub>1</sub>                  | 11    | 30   | 20   | 25   | 5                | 12    | 29    | 9     | 23    | 3     | 8     |
| M <sub>2</sub>                  | 24    | 14   | 4    | 27   | 17               | 13    | 28    | 26    | 6     |       |       |
| M <sub>3</sub>                  | 21    | 1    | 10   | 15   | 22               | 2     | 19    | 7     | 18    | 16    |       |
| machine                         |       |      |      |      | completion times |       |       |       |       |       |       |
| M <sub>1</sub>                  | 29.2  | 53.2 | 77.2 | 84.2 | 91.2             | 95.5  | 99.0  | 102.5 | 115.6 | 128.7 | 141.7 |
| M <sub>2</sub>                  | 22.7  | 45.5 | 68.2 | 88.2 | 108.2            | 121.4 | 134.4 | 139.6 | 144.9 |       |       |
| M <sub>3</sub>                  | 29.2  | 58.5 | 82.5 | 89.5 | 93.7             | 98.0  | 101.5 | 121.5 | 134.5 | 139.7 |       |
| C <sub>max</sub> =              | 144.9 |      |      |      |                  |       |       |       |       |       |       |
| ■ Jobs that use Greenwich glaze |       |      |      |      |                  |       |       |       |       |       |       |

Table 12. Schedule RS obtained after Left-shifting rescheduling

| machine      | jobs' sequence   |      |      |      |       |       |       |       |       |       |  |
|--------------|------------------|------|------|------|-------|-------|-------|-------|-------|-------|--|
| $M_1$        | 11               | 30   | 20   | 25   | 5     | 29    | 9     | 23    | 3     | 8     |  |
| $M_2$        | 24               | 14   | 4    | 27   | 17    | 13    | 28    | 26    | 6     |       |  |
| $M_3$        | 21               | 1    | 10   | 15   | 19    | 7     | 18    | 16    |       |       |  |
| machine      | completion times |      |      |      |       |       |       |       |       |       |  |
| $M_1$        | 29.2             | 53.2 | 77.2 | 84.2 | 91.2  | 94.7  | 98.2  | 111.3 | 124.4 | 137.4 |  |
| $M_2$        | 22.7             | 45.5 | 68.2 | 88.2 | 108.2 | 121.4 | 134.4 | 139.6 | 144.9 |       |  |
| $M_3$        | 29.2             | 58.5 | 82.5 | 89.5 | 93    | 113   | 126   | 131.2 |       |       |  |
| $C_{\max} =$ | 144.9            |      |      |      |       |       |       |       |       |       |  |

Table 13. Schedule RS obtained after applying building new schedules method

| machine      | jobs' sequence   |      |      |      |       |       |       |       |       |       |
|--------------|------------------|------|------|------|-------|-------|-------|-------|-------|-------|
| $M_1$        | 11               | 30   | 20   | 25   | 5     | 19    | 7     | 28    | 26    | 16    |
| $M_2$        | 24               | 14   | 4    | 27   | 17    | 13    | 18    | 6     |       |       |
| $M_3$        | 21               | 1    | 10   | 15   | 29    | 9     | 23    | 3     | 8     |       |
| machine      | completion times |      |      |      |       |       |       |       |       |       |
| $M_1$        | 29.2             | 53.2 | 77.2 | 84.2 | 91.2  | 94.7  | 114.7 | 127.7 | 132.9 | 138.2 |
| $M_2$        | 22.7             | 45.5 | 68.2 | 88.2 | 108.2 | 121.3 | 134.3 | 139.6 |       |       |
| $M_3$        | 29.2             | 58.5 | 82.5 | 89.5 | 93    | 96.5  | 109.6 | 122.7 | 135.7 |       |
| $C_{\max} =$ | 139.6            |      |      |      |       |       |       |       |       |       |

Table 14. Added jobs

| job $J_j$ | pattern       | item       | unit time (hours) | initial processing times $p_k$ (hours) | $M_1$ | $M_2$ | $M_3$ |
|-----------|---------------|------------|-------------------|----------------------------------------|-------|-------|-------|
| 31        | Regency green | Large Jug  | 4                 | 8                                      | ✓     | ×     | ✓     |
| 32        | Marrakesh     | Sauce Boat | 6                 | 6                                      | ✓     | ✓     | ✓     |

Table 15. Schedule RS obtained using Left-shifting rescheduling with two new jobs  $J_{31}$  and  $J_{32}$ 

| machine      | jobs' sequence   |      |      |      |       |       |       |       |       |       |       |
|--------------|------------------|------|------|------|-------|-------|-------|-------|-------|-------|-------|
| $M_1$        | 11               | 30   | 20   | 25   | 5     | 29    | 9     | 23    | 3     | 8     | 32    |
| $M_2$        | 24               | 14   | 4    | 27   | 17    | 13    | 28    | 26    | 6     |       |       |
| $M_3$        | 21               | 1    | 10   | 15   | 19    | 7     | 18    | 16    | 31    |       |       |
| machine      | completion times |      |      |      |       |       |       |       |       |       |       |
| $M_1$        | 29.2             | 53.2 | 77.2 | 84.2 | 91.2  | 94.7  | 98.2  | 111.3 | 124.4 | 137.4 | 144.9 |
| $M_2$        | 22.7             | 45.5 | 68.2 | 88.2 | 108.2 | 121.4 | 134.4 | 139.6 | 144.9 |       |       |
| $M_3$        | 29.2             | 58.5 | 82.5 | 89.5 | 93    | 113   | 126   | 131.2 | 149.2 |       |       |
| $C_{\max} =$ | 149.2            |      |      |      |       |       |       |       |       |       |       |

Table 16. Schedule RS obtained using Building new schedules with two new jobs  $J_{31}$  and  $J_{32}$ 

| machine      | jobs' sequence   |      |      |      |       |       |       |       |       |       |       |       |
|--------------|------------------|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|
| $M_1$        | 11               | 30   | 20   | 25   | 5     | 29    | 19    | 9     | 7     | 13    | 18    | 6     |
| $M_2$        | 24               | 14   | 4    | 27   | 17    | 32    | 3     | 8     |       |       |       |       |
| $M_3$        | 21               | 1    | 10   | 15   | 31    | 23    | 28    | 26    | 16    |       |       |       |
| machine      | completion times |      |      |      |       |       |       |       |       |       |       |       |
| $M_1$        | 29.2             | 53.2 | 77.2 | 84.2 | 91.2  | 94.7  | 98.2  | 101.7 | 121.7 | 134.8 | 147.8 | 153.1 |
| $M_2$        | 22.7             | 45.5 | 68.2 | 88.2 | 108.2 | 124   | 137.1 | 150.1 |       |       |       |       |
| $M_3$        | 29.2             | 58.5 | 82.5 | 89.5 | 115.5 | 128.6 | 141.6 | 146.9 | 152.1 |       |       |       |
| $C_{\max} =$ | 153.1            |      |      |      |       |       |       |       |       |       |       |       |

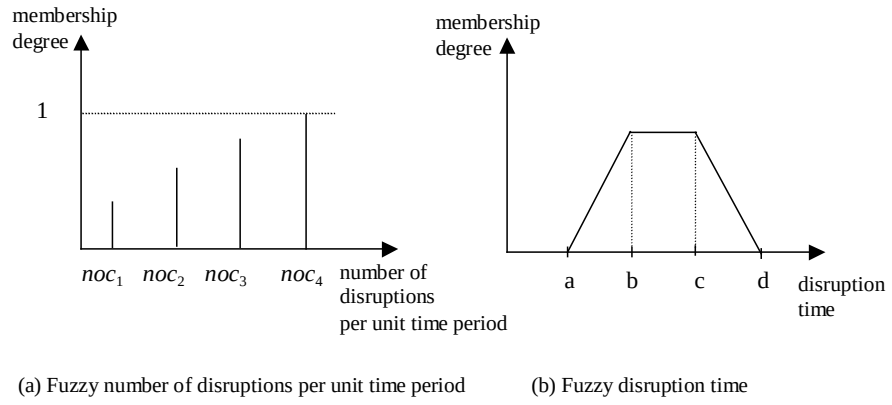


Figure 1. Fuzzy sets that represent an uncertain disruption

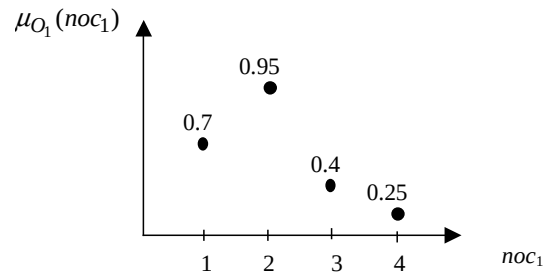


Figure 2. Fuzzy set  $O_1$  that represents an imprecise number of disruption occurrences per unit time period

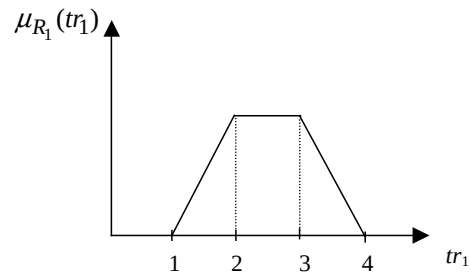


Figure 3. Fuzzy set  $R_1$  that represents an imprecise repair duration

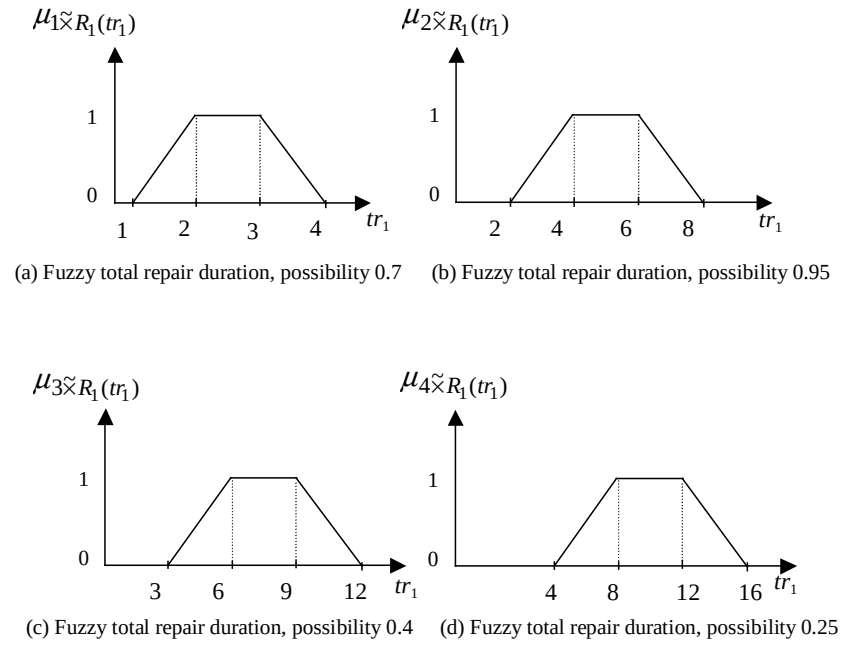


Figure 4. The fuzzy values of the total disruption's repair time for Regency green glaze and the associated possibilities

# **Job-shop Scheduling**

Abstracts

# APPROXIMATE SOLUTIONS FOR A NEW VERSION OF THE CAR SEQUENCING PROBLEM (ABSTRACT)

B. Adenso-Díaz<sup>1</sup>, J. Bautista<sup>2</sup>, J. Pereira<sup>3</sup>

<sup>1</sup>Engineering School at Gijón, Universidad de Oviedo, Spain, [adenso@epsig.uniovi.es](mailto:adenso@epsig.uniovi.es)

<sup>2,3</sup>Industrial Engineering School, Universitat Politècnica de Catalunya, Spain, [joaquin.bautista@upc.es](mailto:joaquin.bautista@upc.es)

**Abstract:** This paper presents a solution procedure for a new variant of the Car Sequencing Problem (CSP), called xCSP. The aim is to satisfy the typical soft-constraints of the CSP while concentrating the maximum possible number of cars with specific options at specific times of the day in order to satisfy other production requirements. Additional constraint ratios are likewise considered that force at least a minimum specific number of consecutive options. The xCSP is formalized and computational results are presented proving the good performance of a GRASP procedure defined for it.

**Key words:** Meta-heuristic Search, Shop-Floor Scheduling, Car Sequencing Problem

## 1. INTRODUCTION

A variant of the sequencing problem of units in mixed product assembly lines is the so-called Car-Sequencing Problem (CSP). The problem, defined by Parello et al. (1986), consists in establishing the entry order onto an assembly line of a set of units (cars) of different types. Each unit of a specific type requires a subset of options that normally require an extra application time at the dedicated workstation.

Each option is therefore subject to a set of maximum load constraint ratios of the form  $p_j/q_j$ , where  $p_j$  is the maximum number of times that option

$j$  may appear in each production sequencing segment of length  $q_j$ . In its original version, the CSP is a feasibility problem; i.e. given an instance of the problem, the aim is to find a sequencing of vehicles (if this exists) that satisfies all the maximum load constraints.

This problem, which has been shown to be NP-hard in the strong sense (Kis, 2004), has been mainly addressed by and received special attention from the scientific community devoted to Constraint Programming, CLP and has become the standard example of software demonstration in Constraint Satisfaction Problem solver manuals.

A natural extension of the CSP problem is proposed in the present paper. The aim of this extension is to: (1) incorporate into the obtained solutions certain properties that are desirable in real shops for manufacturing in a JIT context (regularity in workloads and component consumption), and (2) penalize to a different degree the possible violation of constraints, which is not permitted in a feasibility problem.

On the one hand, a set of additional  $r_j \setminus s_j$  minimum load constraints is incorporated into the original problem that represent the constraint that the minimum number of times that option  $j$  may appear in each production sequencing segment of length  $s_j$  is  $r_j$ . For example, in the binary version of CSP, a constraint of the type  $1 \setminus S$  would indicate that, for such an option, there cannot be  $S$  consecutive vehicles without at least one of these taking this option.

This paper proposes, on the other hand, the conversion of the feasibility problem into an optimization problem by incorporating an objective function that will depend on the slackness variables of the maximum and minimum load constraints.

## 2. MODELING THE PROBLEM

Given an instance of the *Car-Sequencing Problem*  $(C, P, D, d_i, c_{ji}, p_j/q_j)$ , let us call  $C$  and the number of different possible options (air conditioning, sun-roof, etc);  $P$  the number of different car types (each with its own options);  $D$  (total demand) the total number of cars to sequence in the considered time horizon,  $d_i$  (partial demand) the number of cars of type  $i=1..P$  to sequence,  $\sum d_i = D$ ;  $c_{ji}$  being the parameter that indicates the number of units of option  $j=1..C$  required by a car of type  $i=1..P$  (in the binary version of the CSP,  $c_{ji} \in \{0,1\}$ ), and  $p_j/q_j$  the constraint ratios that, for each option  $j=1..C$ , indicate

how many times this option may appear at the most in any sequencing of  $q_j$  vehicles. the basic model for the CSP may be defined by means of expressions (1)-(4):

$$\sum_{i=1}^P x_{it} = 1 \quad \forall t = 1..D \quad (1)$$

$$\sum_{t=1}^D x_{it} = d_i \quad \forall i = 1..P \quad (2)$$

$$\sum_{i=1}^P \sum_{k=l_j(t)}^t c_{ji} x_{ik} \leq \min\{t, p_j\} \quad \forall j = 1..C; \forall t = 1..D; l_j(t) = \max\{1, t+1-q_j\} \quad (3)$$

$$x_{it} \in \{0,1\} \quad \forall i = 1..P; \forall t = 1..D \quad (4)$$

## 2.1 xCSP: The Extended CSP model

On certain occasions to add constraints that force the carrying out of a minimum number of operations until a certain instant. This gives rise to a new group of soft-constraints, which we represent as  $r_j \setminus s_j$ , and which indicate that for each sequencing of  $s_j$  vehicles, option  $j$  has to be present at least  $r_j$  times. These constraints may be established by means of expressions (5).

$$\sum_{i=1}^P \sum_{k=h_j(t)}^t c_{ji} x_{ik} \geq \min\{t, r_j\} \quad \forall j = 1..C; \forall t = 1..D; h_j(t) = \max\{1, t+1-s_j\} \quad (5)$$

where  $h_j(t) = \max\{1, t+1-s_j\}$  is the commencement instance of sequencing segment  $[h_j(t), t]$ ,  $t=1..D$ , of length equal to or less than  $s_j$ .

We define variables:

$$y_{jt}(U) = \left[ \sum_{k=l_j(t)}^t n_{j,u(k)} - \min\{t, p_j\} \right]^+ \quad \forall j = 1..C; \forall t = 1..D; l_j(t) = \max\{1, t+1-q_j\} \quad (6)$$

$$z_{jt}(U) = \left[ \min\{t, p_j\} - \sum_{k=l_j(t)}^t n_{j,u(k)} \right]^+ \quad \forall j = 1..C; \forall t = 1..D; l_j(t) = \max\{1, t+1-q_j\} \quad (7)$$

$$v_{jt}(U) = \left[ \sum_{k=h_j(t)}^t n_{j,u(k)} - \min\{t, r_j\} \right]^+ \quad \forall j = 1..C; \forall t = 1..D; h_j(t) = \max\{1, t+1-s_j\} \quad (8)$$

$$w_{jt}(U) = \left[ \min\{t, r_j\} - \sum_{k=h_j(t)}^t n_{j,u(k)} \right]^+ \quad \forall j = 1..C; \forall t = 1..D; h_j(t) = \max\{1, t+1-s_j\} \quad (9)$$

Note that any sequencing  $U=u(1)..u(D)$  that presents  $y_{jt}(U)=0$  for every option  $j=1..C$  and for every instant  $t=1..D$  of the sequencing, is the solution of the CSP in its original version.

Defining some weights  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$  respectively for each of these new variables, we obtain the final model for xCSP:

$$\min \sum_{t=1}^D \sum_{j=1}^C (\alpha_{jt} y_{jt} + \beta_{jt} z_{jt} + \gamma_{jt} v_{jt} + \delta_{jt} w_{jt}) \quad (10)$$



s.t. equations (1), (2) and (4) and:

$$z_{jt} - y_{jt} + \sum_{i=1}^P \sum_{k=l_j(t)}^t c_{ji} x_{ik} = \min\{t, p_j\} \quad \forall j = 1..C; \forall t = 1..D; l_j(t) = \max\{1, t+1-q_j\} \quad (11)$$

$$w_{jt} - v_{jt} + \sum_{i=1}^P \sum_{k=h_j(t)}^t c_{ji} x_{ik} = \min\{t, r_j\} \quad \forall j = 1..C; \forall t = 1..D; h_j(t) = \max\{1, t+1-s_j\} \quad (12)$$

$$y_{jt}, z_{jt}, v_{jt}, w_{jt} \geq 0 \quad \forall j = 1..C; \forall t = 1..D \quad (13)$$

### 3. A GRASP APPROACH

The complexity of the xCSP model and the practical interest of its solution in a reasonable time on the part of car manufacturers suggest the definition of heuristic procedures that are capable of providing acceptably good solutions with a low computational effort. In this paper we define a GRASP metaheuristic. With the purpose of testing its capability to solve xCSP instances, we have tested the algorithm on 79 instances published on the public library CSPLib, once adapted to the new constraints of the xCSP.

An appropriate experimental framework was defined (considering different weights in eq. (10) and different lengths for the GRASP candidate list), and the corresponding experiments were carried out. After analyzing the results, we can state that our algorithm is able to solve hardly all the instances for the new problem, using very reasonable computational times (around one second of CPU).

### REFERENCES

- Kis, T. (2004) "On the complexity of the car sequencing problem", *Operations Research Letters*, 32: 331-335.
- Parelo, B.D., W.C. Kabat and L. Wos (1986) "Job-shop scheduling using automated reasoning" *Journal of Automated Reasoning*, 2, 1: 1-42.

# **Local Search Methods**

Full Papers

# RANDOM WALKS AND NEIGHBORHOOD BIAS IN OVERSUBSCRIBED SCHEDULING

Mark Roberts, L. Darrell Whitley, Adele E. Howe, Laura Barbulescu  
*Colorado State University*

**Abstract** This paper presents new results showing that a very simple stochastic hill climbing algorithm is as good or better than more complex metaheuristic methods for solving an oversubscribed scheduling problem: scheduling communication contacts on the Air Force Satellite Control Network (AFSCN). The empirical results also suggest that the best neighborhood construction choices produce a search that is largely a greedy random walk of the graph induced by the complete neighborhood.

**Keywords:** Local Search, Heuristic Search, Real World Scheduling

## 1. INTRODUCTION

Local search and metaheuristic search methods that are used in conjunction with local search, such as Tabu search and simulated annealing, have proven to be both robust and effective across a wide range of combinatorial optimization problems. Yet our understanding of local search is largely intuitive: modifications to good solutions are likely to lead to the discovery of other good solutions. No Free Lunch proofs show that in the general case this intuition is wrong; local search is no better than random search over all possible functions (Wolpert and Macready, 1995). Of course, we generally assume (without proof) that the problems where local search is found to be effective have some inherent structure that is being leveraged by local search.

Another common assumption is that carefully crafted neighborhoods are generally better than more random neighborhoods. The choice of neighborhood is always a compromise. The neighborhoods need enough connectivity to assure that the space between random starting points and optima can be traversed in a reasonable number of steps. Over all possible functions, the expected number of local optima is determined by neighborhood size. For a search space of  $S$  points and a neighborhood of size  $k$ , the expected number of local optima for a randomly chosen function is  $S/k + 1$  (Whitley et al., 1997). Large neighborhoods result in few local optima in expectation. However, too large a neighborhood may mean too much time spent deciding on each step. Given  $n$  tasks to schedule, an  $\mathcal{O}(n^2)$  neighborhood can be very costly to evaluate if one is using steepest descent local search.

This paper examines the effect of neighborhood choice on the performance of local search on a large real world application: scheduling the Air Force Satellite Control Network (AFSCN) (Barbulescu et al., 2004b). Approximately 500 contacts with earth orbiting satellites from a set of 16 antennas are scheduled in each 24 hour period. This problem has been shown to be  $\mathcal{NP}$ -complete (Barbulescu et al., 2004b). For the AFSCN scheduling problem with 500 tasks, the shift neighborhood is of size  $\frac{500(499)}{2} = 124,750$ . Yet, other methods, such as genetic algorithms and squeaky-wheel optimization (Joslin and Clements, 1999), find good solutions using less than 50,000 evaluations. Steepest descent local search with this neighborhood configuration is simply not competitive.

This paper examines the bias found in four variations on the shift neighborhood under next descent local search. We consider combinations of two binary characteristics (see Table 1): size and order. For

| Size \ Order | Structured                         | Unstructured                    |
|--------------|------------------------------------|---------------------------------|
| Full         | <b>N1</b> : Ordered                | <b>N2</b> : Random Unrestricted |
| Restricted   | <b>N3</b> : Interaction Restricted | <b>N4</b> : Random Restricted   |

Table 1. The neighborhood types discussed in this paper.

size, neighborhoods are either complete (the full  $\mathcal{O}(n^2)$  neighborhood) or are restricted. For order, the neighborhoods are either structured in some way or they are random samples of larger neighborhoods. The restricted and structured neighborhood (N3) is designed to exploit task interactions that are known to impact schedule quality.

The results on 12 days of actual data show that local search with the right choice of neighborhood is just as effective as a genetic algorithm or squeaky-wheel optimization. What is more surprising is that the normal intuitions about what is the right choice of neighborhood do not hold for AFSCN scheduling. Randomly constructed neighborhoods drawn from the unrestricted full neighborhood yield the best performance. Furthermore, the best local search method appears to be nothing more than a greedy random walk of the graph induced by the search neighborhood.

## 2. AFSCN SCHEDULING

AFSCN scheduling consists of scheduling communication requests for earth orbiting satellites from a set of 16 antennas at 9 ground-based tracking stations. Customers submit requests that are scheduled by humans in a complex arbitration process. This problem is an instance of a single machine problem with release and due dates where the objective is to minimize the number of late jobs, denoted  $1|r_j|\sum U_j$  in the machine scheduling literature (Pinedo, 2002). A formal specification and proof for  $\mathcal{NP}$ -completeness of this problem are found in (Barbulescu et al., 2004b). We provide here a quick introduction of the relevant problem characteristics.

Although AFSCN starts as an oversubscribed scheduling problem, all jobs are eventually scheduled through negotiating relaxed task requirements; so our automated scheduler reduces the effort of the human schedulers by minimizing one of two objective functions: 1) the total number of tasks in conflict or 2) the sum of overlaps between conflicting tasks.

In the current formulation, satellites are grouped according to their orbits: low-altitude and high-altitude. Figure 1 depicts an exemplar request for each altitude. Low-altitude requests (top) typically have short visibility windows (15 minutes) during which a single contact request can be scheduled; these tasks usually have few scheduling alternatives. In contrast, high-altitude requests (bottom) have longer durations (20 minutes or more) with much larger visibility windows. The scheduling alternatives can include different ground tracking stations. Both request types include information about which ground stations and times are possible alternatives.

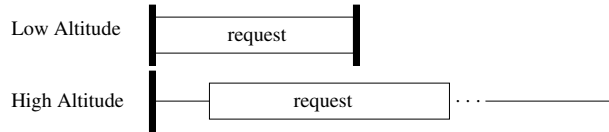


Figure 1. An idealized example of low-altitude (top) and high-altitude (bottom) requests in AFSCN. Low-altitude requests have short visibility windows (15 minutes) with few alternative resources. High-altitude requests have much larger visibility windows and are longer (20 minutes or more); these requests often have many alternative resources.

| ID    | Date     | Size | # Low | # High | Best Conflicts | Best Overlaps |
|-------|----------|------|-------|--------|----------------|---------------|
| Day1  | 10/12/92 | 322  | 153   | 169    | 8              | 104           |
| Day2  | 10/13/92 | 302  | 137   | 165    | 4              | 13            |
| Day3  | 10/14/92 | 311  | 146   | 165    | 3              | 28            |
| Day4  | 10/15/92 | 318  | 142   | 176    | 2              | 9             |
| Day5  | 10/16/92 | 305  | 142   | 163    | 4              | 30            |
| Day6  | 10/17/92 | 299  | 144   | 155    | 6              | 45            |
| Day7  | 10/18/92 | 297  | 142   | 155    | 6              | 46            |
| Mar07 | 03/07/02 | 483  | 225   | 258    | 42             | 773           |
| Mar20 | 03/20/02 | 457  | 194   | 263    | 29             | 486           |
| Mar26 | 03/26/03 | 426  | 183   | 243    | 17             | 250           |
| Apr02 | 04/02/03 | 431  | 185   | 246    | 28             | 725           |
| May02 | 05/02/03 | 419  | 178   | 241    | 12             | 146           |

Table 2. Problem characteristics for the 12 days of AFSCN data used in our experiments. *ID* is used to identify the instance throughout the paper. *Size* is the number of requests in the problem. *# Low* and *# High* are the number of low and high-altitude requests in each problem. *Best conflicts* and *best overlaps* are the best known values for each problem for these two objective functions. The best value for Mar07 is a new best value (the prior best was 774).

Our AFSCN dataset consists of 12 days of real data identified by their dates. Table 2 shows characteristics for these problem instances. The seven older days of data are smaller problems that are easily solved by most of the approaches we have tried. The five newer days are substantially larger problems that are more difficult.

### 3. NEIGHBORHOOD SEARCH

We encode potential solutions using a permutation  $\pi$  of the  $n$  task IDs,  $[1..n]$ . A *schedule builder* is used to generate solutions from the permutation. In effect, the permutation  $\pi$  acts as a priority queue, and the schedule builder places task requests in the schedule based on the order that they appear in  $\pi$ . Each task request is assigned to the first available resource from its list of alternatives and at the earliest possible starting time. This assignment treats the list of alternatives as a rank order, although the actual ordering is arbitrary.

When minimizing the number of conflicts, if the request cannot be scheduled on any of the alternative resources, it is dropped from the schedule (i.e., bumped). When minimizing the sum of overlaps, if a request cannot be scheduled without conflict on any of the alternative resources, it is placed so as to create the minimal overlap with previously scheduled requests. The last two columns of Table 2 show the best-known values for both evaluation functions.

We implemented a next-descent hill-climber that employs the *shift* operator; we accept new solutions that are better or equally good. From a current solution  $\pi$ , a neighborhood is defined by considering all  $(n-1)^2$  pairs  $(x, y)$  of positions in  $\pi$ , subject to the restriction that  $y \neq x-1$ . The neighbor  $\pi'$  corresponding to the position pair  $(x, y)$  is produced by *shifting* the job at position  $x$  into position  $y$ , while leaving all other relative job orders unchanged. If  $x < y$ , then  $\pi' = \text{SHIFT}(\pi, x, y) = (\pi(1), \dots, \pi(x-1), \pi(x+1), \dots, \pi(y), \pi(x), \pi(y+1), \dots, \pi(n))$ . If  $x > y$ , then  $\pi' = \text{SHIFT}(\pi, x, y) = (\pi(1), \dots, \pi(y-1), \pi(x), \pi(y), \dots, \pi(x-1), \pi(x+1), \dots, \pi(n))$ . The pseudocode for the hill-climber and neighborhood variants are shown in Figure 2.

#### 3.1 Complete Neighborhoods for AFSCN

Due to the discrete nature of the evaluation functions and the influence of the schedule builder, most of the options in the shift neighborhood are equivalent – steps on a plateau. Approximately

```

N1-HILL-CLIMBER( $\pi$ )
  for num-evals from 1 to 50000
    do  $x \leftarrow \text{RANDOM}(N)$ 
     $\pi' \leftarrow \pi$ 
    for  $y$  from 0 to  $N - 1$ 
      do if  $x = y$  or  $x = y - 1$ 
        then continue
       $\tau \leftarrow \text{SHIFT}(\pi, x, y)$ 
      if  $\text{EVAL}(\tau) \leq \text{EVAL}(\pi')$ 
        then  $\pi' \leftarrow \tau$ 
  return  $\pi'$ 

N2-HILL-CLIMBER( $\pi$ )
  for num-evals from 1 to 50000
    do  $x \leftarrow y \leftarrow 0$ 
    while  $x = y$  or  $x = y - 1$ 
      do  $x \leftarrow \text{RANDOM}(N)$ 
       $y \leftarrow \text{RANDOM}(N)$ 
     $\pi' \leftarrow \text{SHIFT}(\pi, x, y)$ 
    if  $\text{EVAL}(\pi') \leq \text{EVAL}(\pi)$ 
      then  $\pi \leftarrow \pi'$ 
  return  $\pi$ 

RESTRICTED-HILL-CLIMBER( $\pi, G$ )
  for num-evals from 1 to 50000
    do  $x \leftarrow y \leftarrow 0$ 
    while  $x = y$  or  $x = y - 1$ 
      do  $x \leftarrow \text{RANDOM}(N)$ 
       $y \leftarrow \text{POS}(\pi, \text{R-ADJ}(x, G))$ 
     $\pi' \leftarrow \text{SHIFT}(\pi, x, y)$ 
    if  $\text{EVAL}(\pi') \leq \text{EVAL}(\pi)$ 
      then  $\pi \leftarrow \pi'$ 
  return  $\pi$ 

```

Figure 2. Pseudocode for each of the algorithms we use in this paper. N1-HILL-CLIMBER selects a random  $x$  position and shifts it into all other possible positions. N2-HILL-CLIMBER selects random  $x$  and  $y$  positions. RESTRICTED-HILL-CLIMBER selects a random  $x$  and chooses  $y$  from the tasks adjacent to vertex  $x$  in the interaction graph  $G$ . In this pseudocode,  $\text{RANDOM}(N)$  returns an random integer uniformly from  $(0, N - 1)$ ,  $\text{R-ADJ}(x, G)$  returns the label of a randomly selected adjacent neighbor of  $x$  in  $G$ , and  $\text{POS}(\pi, \text{value})$  returns the position of  $\text{value}$  in  $\pi$ .

40% of the entire neighborhood results in exactly the same *schedule* (Barbulescu et al., 2004a); closer examination reveals that 60-80% are equal-valued though they are translated into different schedules. Our first intuition was to use a structured, complete neighborhood. The **N1** ordered neighborhood randomly chooses a task (i.e., a permutation position)  $x$ , and then evaluates the neighbors produced by systematically shifting  $x$  into *each* possible  $n - 1$  other positions; if no position is acceptable, another  $x$  is selected without replacement. Unfortunately, although systematic and easy to program, we found this neighborhood performed poorly (Barbulescu et al., 2004a). Further investigation showed a detrimental interaction between the domain and the schedule builder. When a shift produces a poorer evaluation, it usually signals that  $x$  is now blocked by the earlier task and no shift of  $x$  later in the schedule can affect that blockage, but many evaluations may be expended trying. If we count the kinds of moves seen *during* search under N1, more than 80% of the considered changes result in worse evaluations; of the remaining 20% (which constitute the actual moves taken), most are plateau moves (equivalent evaluations) and only a few are actually improving moves. This neighborhood induces a significant negative bias against improving or equal moves.

To mitigate this bias, the **N2** unrestricted neighborhood operator randomly selects both  $x$ , the task to be shifted, and  $y$ , the new position where  $x$  is to be inserted. Search algorithms that use this type of random neighborhood move are called “Stochastic Hill Climbers” because they do not systematically explore a neighborhood (Ackley, 1987). **N2** results in a major performance improvement, producing performance competitive with the best previous solutions.

### 3.2 Restricted Neighborhoods for AFSCN

Restricted “critical path” neighborhoods are key to achieving good performance in job-shop and flow-shop scheduling domains. Given that 40% of shifts result in no change to the schedule in AFSCN, one would expect that restricting the search neighborhood to only the tasks that induce a change would produce more efficient search.

Given a task  $u$  to be moved, the **N3** move operator restricts neighbors to only those tasks that are known to interact with  $u$ . More formally, for tasks  $u$  and  $v$  we define  $\text{interacts}(u, v) = \text{true}$  if, on the same resource,  $r_v \leq r_u \leq d_v$  or  $r_v \leq d_u \leq d_v$ , where  $r$  and  $d$  are the release and due dates, respectively. Given alternative scheduling resources, two tasks interact when they contend on one or more common resources. The “interaction” heuristic bears some resemblance to other contention measures, such as the SumHeight heuristic (Beck et al., 1997), where contention is measured across resources. Interaction uses a similar idea but expresses the pair-wise contention across tasks.

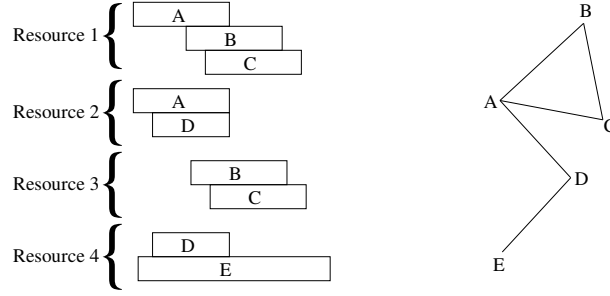


Figure 3. An example of interaction between five requests scheduled on four resources. For this idealized problem, the edge set  $E = \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{D, E\}\}$ .

Task interaction can overestimate the actual amount of contention in the schedule. It includes the entire time window (from release to due date) and disregards processing time. In some situations, it may be possible to schedule both tasks within their respective time windows on the same resource. One of the tasks could also be scheduled on another alternative resource. We calculate pair-wise task interaction for all tasks to build an undirected, unweighted graph where vertices are the tasks and existing edges indicate interaction.

**DEFINITION 1** An interaction graph,  $G$ , is an undirected graph,  $G = (V, E)$ , where the set of vertices,  $V$ , is the set of scheduling tasks and  $E$  consists of edges between vertices  $\{u, v\} \mid u, v \in V, u \neq v, \text{interacts}(u, v) = \text{true}$ .

The interaction graph is designed to provide information about the *potential* conflict between all pairs of tasks in the schedule. Figure 3 shows a simple interaction graph for an idealized oversubscribed problem. Note that interaction is not transitive; it is possible for two tasks that do not interact to both interact with a third task. This case is shown in the example, where tasks A and E both interact with D, but not with each other.

We calculate the interaction graphs for all days of data. The computational cost of calculating the interaction graph is small (less than a second). Figure 4 illustrates the largest connected component

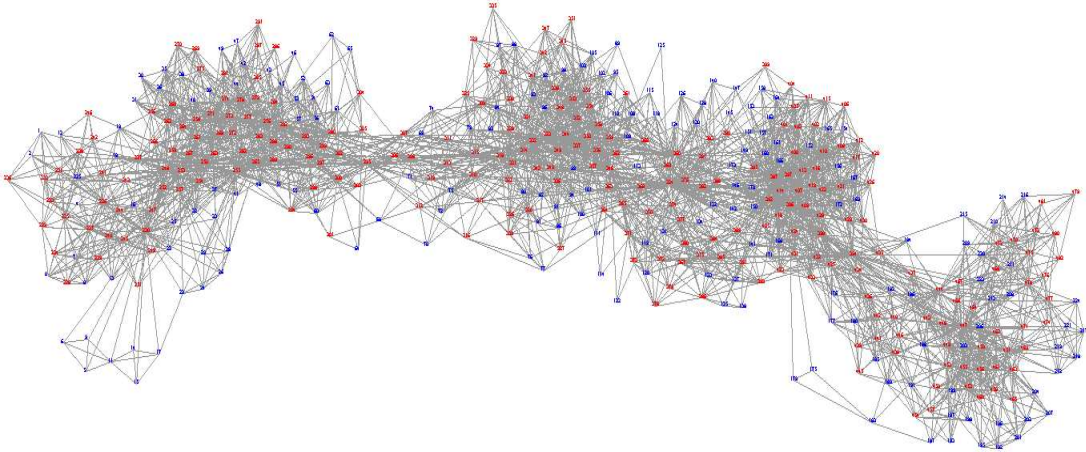


Figure 4. A force-directed layout of the largest connected component in  $G$  for Mar07. The problem contains a single connected component that spans most (92%) of the problem; the remaining tasks have zero degree. High-altitude tasks (in red) are the most connected tasks and are usually in the center. Low-altitude tasks (in blue) are less connected and tend to be along the outside of the graph.

|           | Tasks | Connected | Avg. Degree | $L$  | Tasks of Zero Degree |
|-----------|-------|-----------|-------------|------|----------------------|
| Day1      | 322   | 295       | 6.01        | 4.61 | 27 (0.083)           |
| Day2      | 302   | 273       | 6.31        | 4.61 | 29 (0.096)           |
| Day3      | 311   | 281       | 6.10        | 4.60 | 30 (0.096)           |
| Day4      | 318   | 289       | 6.20        | 4.57 | 29 (0.091)           |
| Day5      | 305   | 274       | 6.20        | 4.61 | 31 (0.102)           |
| Day6      | 299   | 274       | 6.05        | 4.65 | 25 (0.084)           |
| Day7      | 297   | 271       | 6.10        | 7.01 | 26 (0.088)           |
| March0702 | 483   | 440       | 8.32        | 3.80 | 43 (0.089)           |
| March2002 | 457   | 426       | 8.78        | 3.67 | 31 (0.068)           |
| March2603 | 426   | 396       | 7.38        | 4.02 | 30 (0.070)           |
| April0203 | 431   | 396       | 7.14        | 4.96 | 35 (0.081)           |
| May0203   | 419   | 388       | 7.09        | 5.03 | 31 (0.074)           |

*Table 3.* Characterization of  $G$  for each day of AFSCN data. The first column shows the number of tasks in the problem; the second shows the number of tasks connected to the largest connected component. The next two columns show the average degree and the average path length,  $L$ . The last shows the number of zero degree tasks in each problem. The number in parenthesis is the ratio of zero degree tasks in the problem.

of  $G$  for Mar07; this component includes 92% of the vertices. Tasks are roughly in order by ID from left-to-right. Table 3 shows a summary of the largest connected component for all the days of data. The connectedness of these graphs is sparse ( $|E| \ll \binom{n}{2} < \mathcal{O}(n^2)$ ) and shows a low average path length between any two tasks.

The **N3** move operator uses  $G$  to restrict the neighborhood explored by local search; the goal of using **N3** is to focus search on those adjacent new solutions that force a change in the schedule. We iteratively choose a random position  $x$  and shift the task in that position to the position of a randomly selected, interacting neighbor. **N3** dramatically reduces the neighborhood size from  $\mathcal{O}(n^2)$  to the average degree per vertex (see Table 3, column ‘Avg. Degree’). So **N3** also reduces the expected number of neighbors evaluated before selecting a move.

To control for the effects of the structure in the neighborhood, we also implemented a fourth neighborhood: a random, restricted neighborhood. The **N4** move operator creates a random graph with the same degree per vertex as the interaction graph. Each edge in the random restricted graph is randomly connected to another randomly chosen task (excluding itself). **N4** shifts the task in position  $x$  to a randomly selected neighbor in the graph.

## 4. EXPERIMENTS

We first compare the performance of the four neighborhoods. Table 4 shows the final evaluation distributions for 50,000 evaluations over 90 runs. N1 reaches the best-known values much less frequently than the other three neighborhoods, while the other neighborhoods appear equivalent. We focus our analysis on the differences found between the unrestricted and restricted neighborhoods.

Although the minimum values of N2, N3 and N4 are identical, the means and standard deviations vary. Table 4 lists the p-values for the N2 (unrestricted) neighborhood compared to each of the other three neighborhoods using a one-tailed t-test. In nearly all cases, the N2 distributions have lower means and are significantly different than the distributions of the results of other neighborhoods. Thus, the restricted neighborhoods hurt more than they help search for AFSCN.

In addition, informed restriction (N3) does not dominate random restriction (N4). Using a t-test ( $\alpha < .05$ ), N4 is significantly better on Day1, Day2, Day3, and Day5 in both evaluations; it is also better on Apr02 in minimizing conflicts.



|       | N1       |     |       |          | N2  |       |          | N3       |     |       |          | N4       |     |       |          |
|-------|----------|-----|-------|----------|-----|-------|----------|----------|-----|-------|----------|----------|-----|-------|----------|
|       | <i>p</i> | min | $\mu$ | $\sigma$ | min | $\mu$ | $\sigma$ | <i>p</i> | min | $\mu$ | $\sigma$ | <i>p</i> | min | $\mu$ | $\sigma$ |
| Day1  | -        | 9   | 11.07 | 1.32     | 8   | 8.06  | 0.23     | -        | 8   | 9.64  | 0.93     | -        | 8   | 9.19  | 0.98     |
| Day2  | -        | 4   | 5.03  | 1.02     | 4   | 4.00  | 0.00     | -        | 4   | 4.86  | 0.83     | -        | 4   | 4.51  | 0.67     |
| Day3  | -        | 4   | 6.74  | 1.47     | 3   | 3.00  | 0.00     | -        | 3   | 3.36  | 0.61     | -        | 3   | 3.18  | 0.41     |
| Day4  | -        | 4   | 6.99  | 1.55     | 2   | 2.00  | 0.00     | -        | 2   | 2.80  | 0.62     | -        | 2   | 2.98  | 0.65     |
| Day5  | -        | 4   | 6.90  | 1.23     | 4   | 4.09  | 0.29     | -        | 4   | 5.42  | 0.81     | -        | 4   | 5.11  | 0.76     |
| Day6  | -        | 6   | 9.70  | 1.67     | 6   | 6.00  | 0.00     | -        | 6   | 6.29  | 0.46     | -        | 6   | 6.46  | 0.64     |
| Day7  | -        | 6   | 7.91  | 1.12     | 6   | 6.00  | 0.00     | -        | 6   | 6.44  | 0.66     | -        | 6   | 6.31  | 0.51     |
| Mar07 | -        | 53  | 58.02 | 2.55     | 42  | 42.04 | 0.21     | -        | 42  | 42.90 | 0.75     | -        | 42  | 42.87 | 0.78     |
| Mar20 | -        | 32  | 40.27 | 2.70     | 29  | 29.01 | 0.11     | -        | 29  | 29.13 | 0.34     | -        | 29  | 29.29 | 0.46     |
| Mar26 | -        | 21  | 25.94 | 2.14     | 17  | 17.12 | 0.33     | .77      | 17  | 17.09 | 0.29     | .02      | 17  | 17.24 | 0.43     |
| Apr02 | -        | 32  | 36.28 | 2.17     | 28  | 28.00 | 0.00     | -        | 28  | 28.87 | 1.00     | -        | 28  | 28.59 | 0.86     |
| May02 | -        | 14  | 16.79 | 1.49     | 12  | 12.00 | 0.00     | ns       | 12  | 12.00 | 0.00     | *        | 12  | 12.08 | 0.27     |

|       | N1       |      |         |          | N2  |        |          | N3       |     |        |          | N4       |     |        |          |
|-------|----------|------|---------|----------|-----|--------|----------|----------|-----|--------|----------|----------|-----|--------|----------|
|       | <i>p</i> | min  | $\mu$   | $\sigma$ | min | $\mu$  | $\sigma$ | <i>p</i> | min | $\mu$  | $\sigma$ | <i>p</i> | min | $\mu$  | $\sigma$ |
| Day1  | -        | 104  | 172.18  | 33.29    | 104 | 105.73 | 1.49     | -        | 104 | 120.30 | 15.76    | -        | 104 | 112.49 | 8.52     |
| Day2  | -        | 13   | 36.57   | 17.20    | 13  | 13.00  | 0.00     | -        | 13  | 29.72  | 16.63    | -        | 13  | 21.80  | 10.91    |
| Day3  | -        | 35   | 82.22   | 24.86    | 28  | 28.00  | 0.00     | -        | 28  | 31.37  | 6.77     | -        | 28  | 29.43  | 4.17     |
| Day4  | -        | 19   | 64.82   | 27.89    | 9   | 9.13   | 0.72     | -        | 9   | 20.67  | 11.95    | -        | 9   | 20.64  | 10.43    |
| Day5  | -        | 31   | 65.41   | 22.00    | 30  | 30.01  | 0.11     | -        | 30  | 47.13  | 13.65    | -        | 30  | 41.23  | 10.59    |
| Day6  | -        | 50   | 96.98   | 27.60    | 45  | 45.00  | 0.00     | -        | 45  | 49.03  | 8.83     | -        | 45  | 47.31  | 6.58     |
| Day7  | -        | 49   | 87.59   | 25.38    | 46  | 46.00  | 0.00     | -        | 46  | 47.90  | 5.14     | *        | 46  | 47.16  | 3.77     |
| Mar07 | -        | 1173 | 1364.28 | 89.46    | 773 | 778.59 | 7.64     | -        | 773 | 788.43 | 13.30    | -        | 773 | 787.81 | 14.27    |
| Mar20 | -        | 697  | 852.50  | 73.47    | 486 | 495.08 | 6.32     | -        | 486 | 501.36 | 12.51    | -        | 486 | 501.33 | 13.07    |
| Mar26 | -        | 425  | 624.23  | 82.52    | 250 | 258.96 | 25.67    | .38      | 250 | 260.14 | 25.50    | .07      | 250 | 266.22 | 39.11    |
| Apr02 | -        | 958  | 995.26  | 68.72    | 725 | 731.42 | 13.76    | -        | 725 | 754.80 | 29.63    | -        | 725 | 756.96 | 22.54    |
| May02 | -        | 170  | 243.76  | 28.71    | 146 | 146.00 | 0.00     | ns       | 146 | 146.00 | 0.00     | *        | 146 | 146.33 | 1.25     |

*Table 4.* Summary statistics for the final evaluation distributions of conflicts (upper table) and overlaps (lower table). These statistics are taken over 90 runs of 50,000 evaluations each. P-values are computed between unrestricted search and restricted search using a one-tailed t-test that one distribution is lower. High values are written as numbers. Otherwise, a dash indicates significance at the  $\alpha < .0001$  level; a star indicates significance at the  $\alpha < .01$  level; insignificance is marked by ‘ns’.

We conjecture that unrestricted search (N2) converges to the best known values more frequently than restricted search (N3 and N4). To judge this hypothesis, we counted the number of converging and non-converging runs (out of 90) of these three neighborhoods for each day of data. We then performed a  $\chi^2$  test that the proportion of converging runs was the same for N2 as compared to N3 and N4. Mar26 and May02 had similar counts, so the test was not significant. For the other ten days of data, this test revealed that the success of convergence significantly depends on the neighborhood ( $p < .01$ ).

One hypothesis for such variance in the performance of these algorithms is that non-converging runs get stuck on large, suboptimal basins. If this were true, one might expect the final evaluations to be distributed somewhat uniformly above the best-known values. We examined histograms of the final evaluations over 90 runs and found that the non-converging runs end close to the best known values. N2 almost always gets more runs closer, but the difference is still small. For minimizing conflicts, N2 usually gets within one conflict while N3 usually gets within three conflicts. For overlaps, there is slightly more complex behavior. On the seven older days of data, N2 finds solutions within one or two units of overlap while N3 usually gets within 100. On the five new days of data, N2 and N3 closely mimic each others’ final evaluations. Most runs reach within 100 of the best known values.

To assess local differences in neighborhoods, we also examined the number of improving, non-improving, and equal moves under each neighborhood. For the improving moves, we also histogram the change in evaluation. We attempted to correlate these changes in evaluation with specific tasks or task attributes, but found little correspondence of move quality with problem specific information. These results, coupled with the lack of competitive advantage for N3 over N4, lead us to the final conclusion

in our examination of our structured restricted neighborhood: the structured interaction graph provides little advantage over a randomly selected restriction. These graphs do reduce the neighborhood but still remain connected enough such that they can find reasonable solutions.

## 5. SUMMARY AND FUTURE WORK

We examined the effects of problem motivated structure and restricted neighborhood size on the performance of neighborhood operators for a real world scheduling application, AFSCN. Following conventional wisdom, we hypothesized that we could reduce the neighborhood using problem specific structure in a restricted neighborhood. The result was somewhat surprising in that this significantly degraded performance (according to a one-tailed t-test). Search using a restricted neighborhood converges to the best-known values less frequently and shows no major improvement in taking steps that change the evaluation any more than unrestricted search. Moreover, randomly restricted search significantly outperforms structured restricted search for almost half of the problems.

For AFSCN, a restricted neighborhood markedly under-performs an unordered, full neighborhood in next-descent local search. Our evidence suggests that the search is a random walk. We conjecture that search can be modeled as a Markov chain, and we are currently developing a model of the shift neighborhood for unrestricted search. Preliminary results indicate that this model may be quite accurate for AFSCN. We are also extending these analyses to another oversubscribed scheduling domain.

## Acknowledgments

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-03-1-0233. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

We thank Dr. James T. Moore, Associate Professor, Dept. of Operational Sciences, Air Force Institute of Technology and Brian Bayless and William Szary from Schriever Air Force Base for providing the data.

We also thank the anonymous reviewers for their time in critiquing this work and offering suggestions for improvement.

## References

- Ackley, David (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers.
- Barbulescu, L., Howe, A., Whitley, L., and Roberts, M. (2004a). Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem. In *International Conference on Automated Planning and Scheduling (ICAPS-04)*.
- Barbulescu, L., Watson, J., Whitley, D., and Howe, A. (2004b, January). Scheduling space-ground communications for the Air Force satellite control network. *Journal of Scheduling*, 7(1).
- Beck, J.C., Davenport, A.J., Sitarski, E.M., and Fox, M.S. (1997). Texture-based Heuristic for Scheduling Revisited In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*. Providence, RI: AAAI Press, 241–248.
- Joslin, D. E. and Clements, D. P. (1999). "Squeaky Wheel" optimization. In *Journal of Artificial Intelligence Research*, 10, 353–373.
- Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems*. Upper Saddle River, NJ: Prentice Hall.
- Whitley, D., Rana, S., and Heckendorn, R. B. (1997). Representation Issues in Neighborhood Search and Evolutionary Algorithms. In D. Quagliarella and J. Periaux and C. Poloni and G. Winter (Eds.), *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science* (pp. 39-57). New York: Wiley.
- Wolpert, D. H. and Macready, W. G. (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

# **Local Search Methods**

*Abstracts*

# Tabu Search Algorithms For Cyclic Machine Scheduling Problems with blocking

**Peter Brucker , Thomas Kampmeyer\***

Universität Osnabrück, Fachbereich Mathematik/Informatik  
49069 Osnabrück  
{Peter.Brucker, Thomas.Kampmeyer}  
@mathematik.uni-osnabrueck.de

## Abstract

A general framework is developed to model a large variety of different cyclic machine scheduling problems with blocking with the objective to minimize the cycle time. For this framework tabu search algorithms are developed. Neighborhoods are derived which generalize the block-approach based neighborhoods which have been successfully applied to noncyclic job-shop problems and cyclic job-shop problems without blocking.

**keywords:** Local Search, Machine Scheduling

Cyclic scheduling is concerned with the planning of activities that have to be identically repeated at regular intervals. Such types of scheduling problems arise in different application areas like compiler design, manufacturing, digital signal processing, railway scheduling, timetabling, etc. Hanen [2] introduces a general cyclic scheduling problem which among others contains the cyclic job-shop problem as special case. Brucker and Kampmeyer [1] developed tabu-search algorithms for this general problem. Now this general model is extended to cover also blocking situations. This is done by using the alternative graph model [3].

This General Cyclic Scheduling Problem (GCSP) can be described as follows. Let  $T = \{1, \dots, n\}$  be a set of generic tasks or operations (in connection with shop scheduling problems). Task (operation)  $i$  has processing time  $p_i > 0$  and must be performed infinitely often. We denote by  $\langle i; k \rangle$  the  $k$ -th occurrence of the generic task  $i$ . A **schedule** assigns a starting time  $t(i; k)$  to each occurrence  $\langle i; k \rangle$ . A schedule is called **periodic** with cycle time  $\alpha$  if  $t(i; k) = t(i; 0) + \alpha k$  for all  $i \in T$ ,  $k \in \mathbb{Z}$ .

We assume that  $t_i := t(i; 0) \geq 0$  for all  $i \in T$ . A periodic schedule is defined by the vector  $(t_i)_{i \in T}$  and the cycle time  $\alpha$ .

Generalized precedence constraints of the form  $t(i; k) + L_{ij} \leq t(j; k + H_{ij})$  may be given for all arcs  $(i, j) \in E$  of a directed graph  $G = (T, E)$  with vertex set  $T$ .  $L_{ij}$  is called (start-start) **delay** and  $H_{ij}$  is called the **height** of the generalized precedence constraint. Delays are assumed to be nonnegative integers while heights may be arbitrary integers. We also postulate that  $t(i; k) + p_i \leq t(i; k + 1)$  is satisfied for all  $i \in T$ ,  $k \in \mathbb{Z}$ .

Finally, associated with each task  $i$  there is a dedicated machine  $M(i) \in M = \{1, \dots, m\}$ , on which each occurrence  $\langle i; k \rangle$  of  $i$  must be processed.

Consider two operations  $i$  and  $j$  which are processed on the same machine  $m$ . Assume, that both operations are blocking operations and operation  $i$  has a successor  $s(i)$  and  $j$  has a successor  $s(j)$ .

Then we get the following disjunctive constraints:

$$t(s(i); l + H_{i,s(i)}) \leq t(j; k) \vee t(s(j); k + H_{j,s(j)}) \leq t(i; l) \quad (1)$$

---

\*supported by Cusanuswerk

So the following problem must be solved.

$$\begin{aligned}
& \min \alpha & (2) \\
& \text{s.t.} & \\
& t(i; k) = t(i; 0) + \alpha * k & i \in T, k \in \mathbb{Z} & (3) \\
& t(i; k) + L_{ij} \leq t(j; k + H_{ij}) & (i, j) \in E, k \in \mathbb{Z} & (4) \\
& t(i; k) + p_i \leq t(i; k + 1) & i \in T, k \in \mathbb{Z} & (5) \\
& t(s(i); l + H_{i, s(i)}) \leq t(j; k) & \\
& \vee \quad t(s(j); k + H_{j, s(j)}) \leq t(i; l) & i, j \in T \text{ with } i \neq j, \\
& & M(i) = M(j), k, l \in \mathbb{Z} & (6)
\end{aligned}$$

To solve the GCSP we apply tabu-search. We develop several neighborhoods. Some of them are based on a generalization of results for the classical job-shop problem.

Computational tests have been performed by applying our tabu-search to cyclic job-shop and flow-shop problems.

## References

- [1] Brucker, P. and T. Kampmeyer: 2005, ‘Tabu search algorithms for cyclic Machine scheduling problems’. *J. Sched.* **8**, 303–322.
- [2] Hanen, C.: 1994, ‘Study of a NP-hard cyclic scheduling problem: The recurrent job-shop’. *European J. Oper. Res.* **72**, 82–101.
- [3] Mascis, A. and D. Pacciarelli: 2002, ‘Job-Shop scheduling with blocking and no-wait constraints’. *European J. Oper. Res.* **143**, 498–517.

# REPAIR-COST-GUIDED LOCAL SEARCH FOR SCHEDULING

Ignacio Laplagne, Raymond S. K. Kwan, and Ann S. K. Kwan

*School of Computing, University of Leeds, Leeds, LS2 9JT, UK*

{ignacio, rsk, ann}@comp.leeds.ac.uk

**Keywords:** Local Search, infeasible solutions, repair methods, scheduling, driver scheduling

Local Search methods have proved very useful in the field of combinatorial optimisation, particularly in the areas of scheduling and timetabling (Schaerf, 1999). One of the main tasks faced when implementing a local search approach for a specific problem is the design of the neighbourhood structure(s). In particular, it is frequently difficult to find neighbourhoods that are small enough to be computationally feasible to explore, and at the same time allow optimal solutions to be reached by moves inside these neighbourhoods. Local search frameworks generally request that intermediate solutions are *feasible*, i.e. they satisfy all the constraints imposed by the problem definition. Considering infeasible solutions as part of the solution space would often make it easier to define neighbourhood structures in which the optimal solutions are reachable, by allowing a feasible solution to be reached through a sequence of infeasible intermediate solutions; however, there usually is no natural, fair way of comparing feasible and infeasible solutions in terms of the original cost function. Moreover, the search might stray into the infeasible region and never be able to actually output a feasible solution to the user.

Many authors allow for infeasible solutions to be considered by augmenting the original objective function of the problem with a series of *penalty terms* (Hertz, 1992); through them, a solution becomes more unattractive (in terms of the augmented cost function) as the constraints get “more violated”. A drawback of this approach is that the modeller must find the right way of weighing the original cost component against the “amount of constraint violation”, which usually involves empirically determining individual weights for each type of constraint violation that may occur. Some authors provide heuristic methods of adaptively adjusting these weights during the search (Gendreau et al., 1994).

In this paper we present ongoing research on a local search framework that allows infeasible solutions to be part of the search space. This is based on the

use of a fast *repair heuristic*, which transforms infeasible solutions into feasible ones. When considering the cost of an infeasible solution  $s$ , the repair operator is first applied, obtaining a feasible solution  $s_f$ ; the cost of  $s$  is then defined to be the cost of its repaired, feasible equivalent  $s_f$ . Thus, although costs are computed on feasible solutions, the search may proceed on infeasible solutions. Similar ideas, based on the concept of the *Baldwin Effect* (Hinton and Nowlan, 1987), have been developed in the context of evolutionary algorithms (Kaige et al., 2004).

Working within this framework raises a number of questions. How much is the repaired solution an indication of the quality of the original infeasible solution, in terms of its contribution to the search process? What is the behaviour of the search process in the boundary between the feasible and infeasible regions? In particular, how can the transition from a feasible to an infeasible solution be encouraged if the repair heuristic is discouraging it? A number of tests and variants to the method are presented to tackle these questions.

We apply this technique to the driver scheduling problem. Generally modelled as a set covering problem, driver scheduling is the problem of determining the composition of a set of driver shifts (a *schedule*) for a day's transport operation requiring coverage by drivers, while minimising the operational cost of the schedule (Kwan, 2004). This problem is very well suited to this approach for two reasons: first, constraints imposed by labour union rules result in a situation where it is usually difficult to find feasible shifts, and even more so as a result of a "local move"; second, a fast repair heuristic can be easily derived from the generate-and-select approach used to solve set covering problems.

## References

- Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290.
- Hertz, A. (1992). Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35:255–270.
- Hinton, G. E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, 1:495–502.
- Kaige, S., Narukawa, K., and Ishibuchi, H. (2004). Lamarckian repair and darwinian repair in emo algorithms for multiobjective 0/1 knapsack problems. In *GECCO 2004 (late breaking papers)*, Seattle, USA.
- Kwan, Raymond S. K. (2004). Bus and train driver scheduling. In Leung, J. Y-T, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 51. CRC Press.
- Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127.

# **Machine Scheduling**

Full Papers



# BICRITERIA SCHEDULING OF EQUAL LENGTH JOBS WITH RELEASE DATES ON IDENTICAL PARALLEL MACHINES

Hari Balasubramanian, John Fowler, Ahmet Keha  
*Industrial Engineering, Arizona State University*

**Abstract** We consider bicriteria scheduling problems on parallel, identical machines with job release dates. The jobs are assumed to have equal processing times. Our main goal in this paper is to report complexity results for bicriteria problems in this environment when the number of machines is assumed to be a fixed or constant value. The results are based on a straightforward use of the dynamic program of Baptiste (2000). When the the number of machines is given, we show that it is possible to minimize, in polynomial time, a composite linear objective function involving 1) sum of completion times and total tardiness, and 2) under a given condition the sum of weighted completion times and total tardiness. We then use a technique proposed by Aneja and Nair (1979) to generate extreme schedules on the efficient frontier for the first problem and a subset of them for the second one. We also show that it is possible to generate in polynomial time the set of Pareto Optimal points for bicriteria problems with  $C_{max}$  as one of the criteria and either of  $\sum w_j C_j$  and  $\sum T_j$  as the other.

**Keywords:** bicriteria scheduling, parallel machines, equal processing time jobs.

## 1. Introduction

We consider the problem of scheduling  $n$  jobs  $J_1, J_2, \dots, J_n$  on  $m$  parallel, identical machines. Each job has a release date  $r_j$ , a due-date  $d_j$  and weight  $w_j$ . All jobs have the same processing time  $p$ . The goal is to find a completion time  $C_j$  for each job  $j$  to solve some bicriteria optimization problems. The completion times have to be feasible, that is 1) jobs start after their release date i.e.  $\forall j, C_j - p \geq r_j$  and 2) no more than  $m$  machines are used at any time  $t$ , i.e.  $\forall t, |\{J_j | C_j - p \leq t < C_j\}| \leq m$ . Additionally, all schedules are non-preemptive.

Since criteria are often conflicting, it is rare that a single solution is best for all of them. In any multicriteria problem it is important to point out the nature

of optimization being performed given that such trade-offs exist.

Let  $S$  be the set of feasible solutions for a bicriteria optimization problem and  $z_1(X)$  and  $z_2(X)$  be the objective values for criteria  $z_1$  and  $z_2$  (both of which need to be minimized) for a feasible solution  $X \in S$ . We follow, for the most part, the notation and terminology of T'Kindt and Billaut (2002) and Hoogeveen (2004).

It may be of interest to minimize a secondary criterion  $z_2$  given the optimal value of the primary criterion  $z_1$ . This is called lexicographic or hierarchial optimization represented as  $\text{Lex}(z_1, z_2)$ . Sometimes the decision-maker may have a linear function in mind that he intends to minimize: a composite function of the form  $F_l(z_1, z_2) = \alpha * z_1 + (1 - \alpha) * z_2$ , where  $0 \leq \alpha \leq 1$ . But perhaps of most interest - and in many cases the most difficult to generate - is the set of Pareto optimal or non-dominated solutions.

**DEFINITION 1** A solution  $X^*$  is Pareto optimal or non-dominated if there exists no other solution  $X \in S$  for which  $z_1(X) \leq z_1(X^*)$  and  $z_2(X) \leq z_2(X^*)$  where at least one of the inequalities is strict.

**DEFINITION 2** A solution  $X^*$  is said to be weakly Pareto optimal if there exists no other solution  $X \in S$  for which  $z_1(X) < z_1(X^*)$  and  $z_2(X) < z_2(X^*)$

Let  $co(ND)$  represent the convex hull of all the non-dominated solutions when each solution is plotted on in criteria space (with each axis representing a criterion) based on the values the solution has for the two criteria. Hoogeveen (2004) defines the *efficient frontier* as the lower envelope of  $co(ND)$  and an *extreme point* as a Pareto optimal solution that is also a vertex of the efficient frontier. The set of extreme points is of interest as sometimes it is not possible to generate all the Pareto optimal solutions. It is worthwhile to note that there exists an extreme point that is an optimal solution to the linear combination  $F_l(z_1, z_2) = \alpha * z_1 + (1 - \alpha) * z_2$ , for any  $\alpha$  such that  $0 \leq \alpha \leq 1$ . The extreme points are also a subset of the *supported* solutions, defined by T'Kindt and Billaut (2002) as set of Pareto optimal points located on the efficient frontier. The set of *non-supported* solutions are set of set of Pareto optimal points that do not lie on the efficient frontier.

Insert figure 1 here.

In this paper, we consider the minimization of a linear combination of criteria  $F_l(z_1, z_2) = (\alpha * z_1 + (1 - \alpha) * z_2)$ , where  $0 \leq \alpha \leq 1$ , and  $z_1$  and

$z_2$  are the two objectives) for the following problems. In the notation of T'Kindt and Billaut (2002), we have 1)  $P_m|r_j, p_j = p|F_l(\sum C_j, \sum T_j)$  and 2)  $P_m|r_j, p_j = p|F_l(\sum w_j C_j, \sum T_j)$ . In the next step, we generate the extreme schedules on the efficient frontier for the first problem and a subset of the extreme schedules on the efficient frontier for the second one.

We also present algorithms to generate the set of Pareto optimal points for some bicriteria problems. The technique used is the  $\epsilon - constraint$  approach and is written in the  $\alpha|\beta|\gamma$  scheduling notation as  $\epsilon(z_1|z_2)$  (T'Kindt and Billaut (2002)). It implies the minimization of  $z_1$  given a fixed value of  $z_2$ . Our bicriteria problems are represented as:

- 1)  $P_m|r_j, p_j = p|\epsilon(\sum w_j C_j|C_{max})$  and 2)  $P_m|r_j, p_j = p|\epsilon(\sum T_j|C_{max})$ .

## 2. Related Work

The recent survey Hoogeveen (2004) is a comprehensive study of multicriteria scheduling research to date. The survey reveals the dearth of theoretical results in bicriteria problems involving parallel machines, mostly because very few single criteria problems have polynomial time complexities. Recently, however, some results have emerged in this area. Baptiste and Brucker (2004) show that  $P_m||Lex(\sum C_j, \sum U_j)$  (the minimization of the total number of tardy jobs given the optimal value of the total completion time) is solvable in pseudopolynomial time and that  $P_2||Lex(\sum C_j, \sum U_j)$  is NP-hard. Gupta *et al.* (2003) prove further complexity results: they show that  $P||Lex(\sum C_j, C_{max})$  is strongly NP-hard and thus  $P||Lex(\sum C_j, \sum U_j)$  is strongly NP-hard as well;  $P||Lex(\sum C_j, T_{max})$  can be solved in pseudopolynomial time; and  $P_m||Lex(\sum C_j, \sum U_j)$  can be solved in polynomial provided the processing times are all different. In a later paper, Gupta and Ruiz-Torrez (2004) propose heuristic algorithms for generating a set of efficient (non-dominated) schedules on identical parallel machines involving total flow-time and total number of tardy jobs. Sarin and Prakash (2004) propose polynomial algorithms for bicriteria scheduling of various pairs of traditional scheduling objectives of in a parallel machine environment with equal release dates assuming that all jobs have equal processing times. Their approach is lexicographic optimization i.e. they assume a primary measure and a secondary measure. Angel *et al.* (2003) identify a class of multiobjective optimization problems possessing a fully polynomial time approximation scheme (FPTAS) for computing an  $\epsilon$ -approximate Pareto curve. T'kindt *et al.* (2001) consider a bicriteria scheduling problem on unrelated parallel machines applicable to a in the production of glass bottles.

In the last few years, a slew of parallel machine scheduling problems with release dates, whose complexities had previously been unknown, were shown to be polynomial (Baptiste (2000), Baptiste *et al.* (2002)). A key characteristic which makes these problems tractable is the polynomial number of time points at which jobs can start or finish in a schedule. We return to this property later in section 5.

For our bicriteria optimization requirements, we use the the dynamic program of Baptiste (2000) that solves  $\sum w_j C_j$  and  $\sum T_j$  optimally (each criteria individually) and works for sum functions ( $\sum f_j$ ) that are non-decreasing for all  $f_i$  and monotonous for all  $f_i - f_j$ . For notational convenience we call this dynamic program DPB (Dynamic Program by Baptiste).

### 3. Minimizing a composite linear function

As stated earlier, DPB works for functions of the type  $\sum f_j$  that are non-decreasing and monotonous, formally defined below, (Baptiste, 2000):

**DEFINITION 3.1** *The functions  $f_i$  are non-decreasing, i.e.,  $\forall t_1, \forall t_2 > t_1, f_i(t_1) \leq f_i(t_2)$  and the functions  $f_i - f_j$  are monotonous, i.e., either  $\forall t_1, \forall t_2 > t_1, (f_i - f_j)(t_1) \leq (f_i - f_j)(t_2)$  or  $\forall t_1, \forall t_2 > t_1, (f_i - f_j)(t_1) \geq (f_i - f_j)(t_2)$ .*

We now consider  $P_m | r_j, p_j = p | F_l(\sum C_j, \sum T_j)$  scheduling problem.

**PROPOSITION 3.1**  *$f_i - f_j$  is monotonous for any composite linear function of the form  $\alpha * \sum C_j + (1 - \alpha) * \sum T_j$ , where  $0 \leq \alpha \leq 1$ .*

*Proof:* In  $f_i - f_j$ , for any pair of jobs  $i$  and  $j$  at any time point, the  $\alpha * \sum C_j$  part of the composite linear function is always 0. Therefore when the function is compared at any two time points  $t_1$  and  $t_2, t_1 \leq t_2$ , only the  $(1 - \alpha) * \sum T_j$  can be non-zero. Since  $f_i - f_j$  is monotonous for  $(1 - \alpha) * \sum T_j$ , it is monotonous for  $\alpha * \sum C_j + (1 - \alpha) * \sum T_j$  as well.

We continue with the  $P_m | r_j, p_j = p | F_l(\sum w_j C_j, \sum T_j)$  scheduling problem. Here, the linear function has the form:  $\alpha * \sum w_j C_j + (1 - \alpha) * \sum T_j$ . In the analysis that follows, we attempt to characterize the values of  $\alpha$  for which  $f_i - f_j$  is monotonous.

Let  $w_{min} = \min\{w_i - w_j | J_i, J_j, w_i > w_j, d_i > d_j\}$ . Then we have the following proposition.

**PROPOSITION 3.2** *For any values of  $\alpha$  that satisfy  $\alpha \geq 1/(1 + w_{min})$ , the composite linear functions  $f_i - f_j$  involving weighted completion time and tardiness are monotonous.*

*Proof:* Note that for those pairs of jobs for which,  $w_i \geq w_j$  and  $d_i \leq d_j$ ,  $f_i - f_j$  is monotonous. So we focus our attention on pairs of jobs for which  $w_i > w_j$  and  $d_i > d_j$ . We consider six exhaustive cases based on the locations of the time points  $t_1$  and  $t_2$ .

1.  $t_1, t_2 \leq d_i$ :  $f_i - f_j$  is monotonous for any pair of jobs as tardiness of the jobs is zero and only the  $\alpha * \sum w_j C_j$  part can be non-zero.

2.  $t_1, t_2 > d_j$  and  $t_1, t_2 \leq d_i$ : Here for  $f_i - f_j$  to be monotonous,

$$\alpha(w_i - w_j)t_1 - (1 - \alpha)(t_1 - d_j) \leq \alpha(w_i - w_j)t_2 - (1 - \alpha)(t_2 - d_j)$$

$$\alpha * (w_i - w_j)(t_2 - t_1) \geq (1 - \alpha)(t_2 - t_1)$$

$$(1 - \alpha)/\alpha \leq (w_i - w_j)$$

3.  $t_1, t_2 > d_i$ : Here we need,

$$\alpha(w_i - w_j)t_1 + (1 - \alpha)(d_j - d_i) \leq \alpha(w_i - w_j)t_2 + (1 - \alpha)(d_j - d_i)$$

This is automatically ensured as the  $(1 - \alpha)(d_j - d_i)$  is constant at both  $t_1$  and  $t_2$ .

4.  $t_1 \leq d_j$  and  $d_j < t_2 \leq d_i$ : We consider  $t_1 = d_j$  as if we show that  $(f_i - f_j)t_2 \geq (f_i - f_j)d_j$  then from case 1,  $(f_i - f_j)t_2 \geq (f_i - f_j)t_1$  for all  $t_1 < d_j$ . We therefore need,

$$\alpha(w_i - w_j)d_j \leq \alpha(w_i - w_j)t_2 - (1 - \alpha)(t_2 - d_j)$$

which leads us to:

$$(1 - \alpha)/\alpha \leq (w_i - w_j)$$

5.  $d_j < t_1 \leq d_i$  and  $t_2 > d_i$ : For  $f_i - f_j$  to be monotonous,

$$\alpha(w_i - w_j)t_1 - (1 - \alpha)(t_1 - d_j) \leq \alpha(w_i - w_j)t_2 + (1 - \alpha)(d_j - d_i)$$

which leads us to:

$$(1 - \alpha)(d_i - t_1) \leq \alpha(w_i - w_j)(t_2 - t_1)$$

Since  $(t_2 - t_1)$  will tend to equal  $(d_i - t_1)$  when  $t_2$  is close to  $d_i$  we need:

$$(1 - \alpha)/\alpha \leq (w_i - w_j)$$

6.  $t_1 \leq d_j$  and  $t_2 > d_i$ : We consider  $t_1 = d_j$  for the same reason explained in case 4. For  $f_i - f_j$  to be monotonous, we need to satisfy,

$$\alpha(w_i - w_j)d_j \leq \alpha(w_i - w_j)t_2 + (1 - \alpha)(d_j - d_i)$$

which leads us to:

$$(1 - \alpha)(d_i - d_j) \leq \alpha(w_i - w_j)(t_2 - d_j)$$

Since  $(t_2 - d_j)$  will tend to equal  $(d_i - d_j)$  when  $t_2$  is close to  $d_i$  we end up once more with the equation that gives us values of  $\alpha$  that ensure that  $f_i - f_j$  is monotonous:

$$(1 - \alpha)/\alpha \leq (w_i - w_j)$$

The result means that even if the  $w_{min} = 1$ , DPB can be used to optimally solve the  $\alpha * \sum w_j C_j + (1 - \alpha) * \sum T_j$  as long as  $\alpha$  is 0.5 or higher. This conclusion is quite interesting and useful as it implies that the  $\sum w_j C_j + \sum T_j$  function can be solved in polynomial time given the number of machines, while the closely related objective  $\sum w_j T_j$  (which is not monotonous for all  $f_i - f_j$  functions) is still an open problem.

#### 4. Generating Extreme Schedules

Aneja and Nair (1979) generate the extreme points for a bicriteria transportation problem modeled as assignment problem. Though our optimization technique is a dynamic program it is still possible for us to use their algorithm to generate the extreme schedules on the efficient frontier for the bicriteria problem involving total completion time and total tardiness. The steps of the algorithm are given below. For notational convenience we use  $z_1$  for  $\sum C_j$  and  $z_2$  for  $\sum T_j$ .  $z_1^{(k)}$  and  $z_2^{(k)}$  are used to denote the objective values of  $z_1$  and  $z_2$  and point  $k$ .

*Step 0:* Use DPB to find the optimal value  $z_1^{(1)}$  for the  $z_1$  objective. Record the value of  $z_2^{(1)}$  for this solution. Set  $k = 1$ . Similarly use DPB to find the optimal value  $z_2^{(2)}$  for the  $z_2$  objective and record the value of  $z_1^{(2)}$  for this

solution. If  $(z_1^{(1)}, z_2^{(1)}) = (z_1^{(2)}, z_2^{(2)})$ , stop. Set  $k = k + 1$ . Define sets  $L = (1, 2)$  and  $E = \emptyset$  and go to step 1.

*Step 1:* Choose an element  $(r, s) \in L$  and set  $a_1^{r,s} = |z_2^s - z_2^r|$  and  $a_2^{r,s} = |z_1^s - z_1^r|$ . Use the DPB now to minimize:  $a_1^{r,s} * z_1 + a_2^{r,s} * z_2$ . Let  $z_1^*$  and  $z_2^*$  be the values of objectives  $z_1$  and  $z_2$  respectively. If  $(z_1^*, z_2^*)$  is equal either to  $((z_1^{(r)}, z_2^{(r)}))$  or  $(z_1^{(s)}, z_2^{(s)})$ , set  $E = E \cup \{(r, s)\}$  and go Step 2. Else record,  $((z_1^{(k)}, z_2^{(k)}))$  such that  $z_1^{(k)} = z_1^*$  and  $z_2^{(k)} = z_2^*$ . Set  $k = k + 1$ ,  $L = L \cup (r, k)$ ,  $(k, s)$  and go to step 2.

*Step 2:* Set  $L = L - (r, s)$ . If  $L = \emptyset$ , stop. Otherwise go to Step 1.

Insert figures 2 and 3 here.

The algorithm differs from the one given in Aneja and Nair (1979) in the following feature: the points 1 and 2 with which we start the algorithm may not always be extreme points representing  $Lex(\sum C_j, \sum T_j)$  and  $Lex(\sum T_j, \sum C_j)$ . In other words, they may be weakly Pareto optimal solutions. However, it is easy to see that we will still end up with all the undiscovered extreme points including the lexicographic ones. This is because all the undiscovered efficient extreme will lie in halfspace beneath the line joining the two initial points (Figure 2), which represents the new objective function. The proof of correctness and finiteness will follow the same steps as the one described in Aneja and Nair (1979).

For the problem with  $\sum w_j C_j$  and  $\sum T_j$ , however, the same algorithm can only be used to generate a subset of the extreme schedules. The only modification in the algorithm will involve changing the 2nd point in Step 0 of the algorithm to the one obtained by solving  $\alpha * \sum w_j C_j + (1 - \alpha) \sum T_j$ , where  $\alpha = 1/(1 + w_{min})$ . Thus the procedure will generate all the extreme points that lie in halfspace below the line joining the two initial points (Figure 3).

The DPB has a complexity of  $O(n^{3m+4})$  (Baptiste 2000). If the number of extreme points is  $k$ , the complexity of this procedure is  $O(kn^{3m+4})$ .

## 5. Bicriteria Problems involving $C_{max}$

One of the main reasons for the polynomial solvability of release date problems with equal processing times is that the number of time points at which jobs can start or finish in a schedule is bounded by  $O(n^2)$ . The following lemma from Baptiste (2000) expresses this more formally. If  $\zeta$  denotes the schedule, among optimal ones then the following lemma characterizes the time points at which jobs start and end on the schedule  $\zeta$ .

**LEMMA 3** *The time points at which jobs start and end on the schedule  $\zeta$  belong to  $T = \{t : \exists r_i, \exists l \in \{0, \dots, n\}, t = r_i + lp\}$*

This property is particularly useful in bicriteria optimization when one of the criteria is  $C_{max}$ . The optimal value of  $C_{max}$  can be obtained using the algorithm in Desousky et al. (1990). DPB is structured so that the optimal value function  $F_n = ((\min_{t \in T} t, \dots, \min_{t \in T} t), (\max_{t \in T} t, \dots, \max_{t \in T} t))$  is calculated between a least possible time points in  $T$  on the machines and greatest possible time points in  $T$ . To achieve  $Lex(C_{max}, \sum w_j C_j)$  and  $Lex(C_{max}, \sum T_j)$ , we merely need to rewrite the optimal value function as  $F_n = ((\min_{t \in T} t, \dots, \min_{t \in T} t), (C_{max}, \dots, C_{max}))$ .

In order to generate the set of pareto optimal points, we set the optimal value functions to

$$F_n = ((\min_{t \in T} t, \dots, \min_{t \in T} t), (t', \dots, t')), \forall t' \text{ such that } C_{max} \leq t' \leq \max_{t \in T} t.$$

Since the upperbound on the number of values  $t'$  can take is  $O(n^2)$ , the overall complexity of the algorithm is  $O(n^2 n^{3m+4}) = O(n^{3m+6})$ .

## 6. Conclusions and Future Research

In this paper, we consider bicriteria scheduling problems in an identical parallel machine environment where jobs have equal processing times and unequal release dates. We use the framework provided by DPB of Baptiste (2000) and the algorithm of Aneja and Nair (1979) to generate the efficient frontier for  $\sum C_j$  and  $\sum T_j$ , and a subset of the efficient schedules for  $\sum w_j C_j$  and  $\sum T_j$ . The efficient schedules are generated in running time that is polynomial given the number of machines.

We also use DPB to generate the pareto optimal set for bicriteria problems involving  $C_{max}$  and either one of  $\sum w_j C_j$  or  $\sum T_j$ . This algorithm is also polynomial given the number of machines.

Though we happen to discover the lexicographic points while generating the efficient frontier for  $\sum C_j$  and  $\sum T_j$ , we are working to propose computationally more efficient algorithms, especially in cases where  $\sum C_j$  is a primary measure.

Our research is also focussed on bicriteria scheduling problems in the same environment involving other criteria such as  $L_{max}$  and  $\sum U_j$  and  $\sum w_j U_j$  in terms of lexicographic optimization, generation of the efficient frontier and the set of pareto optimal schedules.



## References

- Aneja Y.P. and Nair K.P.K. (1979), Bicriteria Transportation Problem, *Management Science*, 25, 73-78.
- Angel E., Bampis E. and Kononov A. (2003), On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems, *Theoretical Computer Science* 306, 319-338.
- Baptiste P. (2000), Scheduling Equal Length jobs on identical parallel machines, *Discrete Applied Mathematics*, 103, 21-32.
- Baptiste P. , Brucker, P., Knust, S. Timkovsky, V.G. (2004), Ten notes on equal-processing-time scheduling; At the frontiers of solvability in polynomial time, *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2, 111-127.
- Dessousky M.I., Lageweg B.J., Lenstra J.K., van de Velde S.L. (1989), Scheduling identical jobs on uniform parallel machines, *Statist. Neerlandica*, 44, 115-123.
- Gupta J.N.D. and Ruiz-Torrez A.J. , Generating efficient schedules for identical parallel machines involving flow time and tardy jobs, *European Journal of Operational Research*, Article in Press.
- Gupta J.N.D. , Ruiz-Torrez A.J. and Webster S.(2003) ,Minimizing maximum tardiness and number of tardy jobs on parallel machines subject to minimum flow time, *Journal of Operational Research Society*, 54, 1263-1274.
- Hoogeveen H. (2004) , Multicriteria Scheduling, *European Journal of Operational Research*, Article in Press.
- Lin C-H. and Liao C-J. (2004), Makespan minimization subject to flowtime optimality on identical parallel machines, *Computers and Operations Research*, 31, 1655-1666.
- Sarin S. and Prakash, D. (2004), Equal processing time Bicriteria Scheduling on Parallel Machines, *Journal of Combinatorial Optimization*, 8, 227-240.
- T'kindt V., Billaut J-C. and Proust C. (2001), Solving a bicriteria scheduling problem on unrelated parallel machines occurring in the glass bottle industry, *European Journal of Operational Research*, 135, 42-49.
- T'kindt V. and Billaut J-C. (2002), *Multicriteria Scheduling*, Springer, 2002.

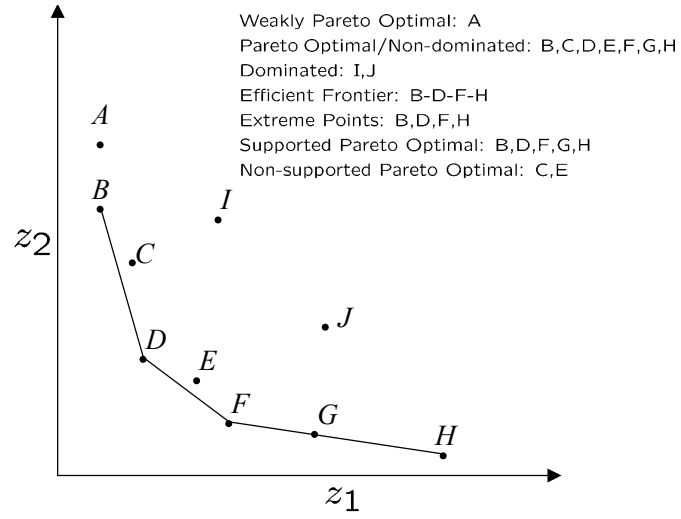


Figure 1. Solutions in criteria space

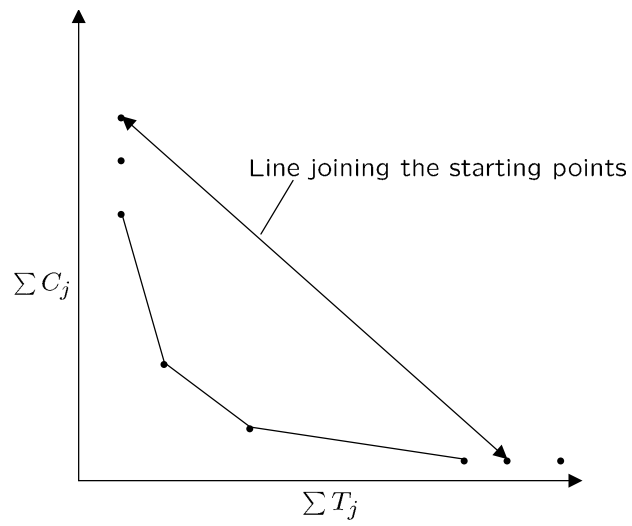


Figure 2. Generating extreme schedules for  $\sum C_j$  and  $\sum T_j$

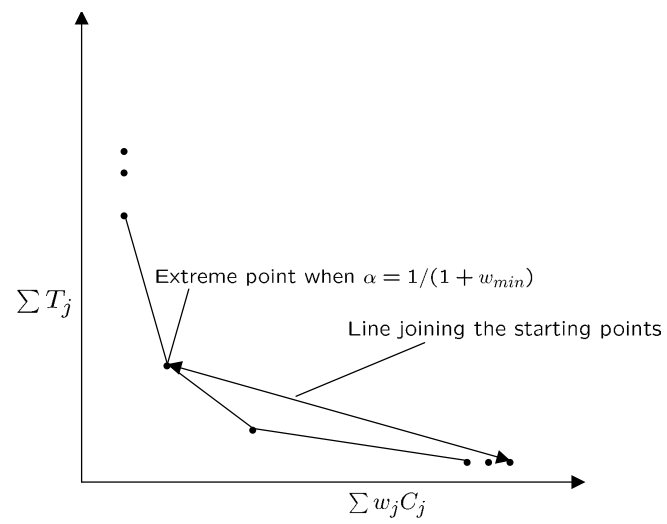


Figure 3. Generating a subset of extreme schedules for  $\sum w_j C_j$  and  $\sum T_j$

# ENUMERATION OF PARETO OPTIMA FOR UNIFORM PARALLEL MACHINE SCHEDULING WITH TWO CRITERIA

K.BOUIBEDE-HOCINE

*Université François-Rabelais de Tours*

*Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France*

karima.hocine@etu.univ-tours.fr

V.T'KINDT

*Université François-Rabelais de Tours*

*Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France*

tkindt@univ-tours.fr

**Abstract** In this paper we consider the solution of an uniform parallel machine scheduling problem with two criteria. The aim is to minimise the makespan as well as the maximum lateness of jobs. We first propose a heuristic by progressive construction which tries to solve the problem without studying all possible solutions. An efficient branch-and-bound algorithm to compute a strict Pareto optimum is presented. In this method we introduce the preemptive feasibility test and energetic test which are used to prune nodes in the search tree. We have also integrated other methods in this branch-and-bound algorithm to reduce the search space.

**Keywords:** Multicriteria Scheduling - Pareto optima - Feasibility.

## 1. Introduction

The scheduling problem that we consider can be stated as follows. Consider  $n$  jobs to be scheduled without preemption on  $m$  uniform parallel machines, such that each machine has its own speed denoted by  $V_j$ ,  $\forall j = 1, \dots, m$ . Each job  $J_i$  has a release time  $r_i$  and a due date  $d_i$ . Job  $J_i$  can be performed by any machine  $M_j$ , thus requiring a processing time  $p_{ij}$  with  $p_{ij} = \frac{p_i}{V_j}$ ,  $\forall i = 1, \dots, n, \forall j = 1, \dots, m$ , where  $p_i$  is the absolute processing time of the job  $J_i$ . The bicriteria scheduling problem

concerns the simultaneous minimisation of the makespan  $C_{max}$  and the maximum lateness  $L_{max}$ . For a given schedule, each job  $J_i$  completing at time  $C_i$ , the two objectives can be computed as follows:

$$C_{max} = \max_{i=1..n} \{C_i\} \text{ and } L_{max} = \max_{i=1..n} \{C_i - d_i\}.$$

It is remarkable that the solution of a multicriteria scheduling problem leads to calculate one or more strict Pareto optima since there rarely exists a single solution that optimises all criteria [T'kindt and Billaut, 2002]. A solution  $s$  is a strict Pareto optimum iff there does not exist another solution  $s'$  such that  $C_{max}(s') \leq C_{max}(s)$  and  $L_{max}(s') \leq L_{max}(s)$  with at least one strict inequality. Accordingly, criteria vector  $[C_{max}(s); L_{max}(s)]$  is said to be strictly non dominated. Notice that often the enumeration of strict Pareto optima is restricted to the enumeration of one solution per strictly non dominated criteria vectors. We denote by  $E$  this restricted set of strict Pareto optima. In this paper we focus on the enumeration of set  $E$  and following the classic three-field notation of scheduling problems introduced in [Graham et al, 1979] and extended to multicriteria scheduling problems in [T'kindt and Billaut, 2002], this problem can be referred to as  $Q/r_i, d_i/C_{max}, L_{max}$ . The calculation of the strict Pareto optima for the problem  $Q/r_i, d_i/C_{max}, L_{max}$  is NP-hard.

Related problems have been tackled in the literature. When only criterion  $L_{max}$  is minimised, the  $Q/r_i/L_{max}$  is NP-hard. The problem  $P/r_i, q_i/C_{max}$  is NP-hard in the strong sens because it is a generalization of the problem  $1/r_i, q_i/C_{max}$  which is NP-hard in the strong sens [Garey and Johnson, 1979]. An exact method has been proposed in [Tercinet, 2004] to solve the problem  $P/r_i, q_i/C_{max}$ .

The remainder is organized as follows. In Section 2 we present the main lines of the enumeration of strict Pareto optima as conducted in this paper. This enumeration is based on the successive solution of decision problems, for which we propose heuristic and exact algorithms in the rest of the paper. A heuristic which tries to solve quickly the problem is proposed in Section 3, whilst infeasibility tests which are based on the relaxation of the initial problem are introduced in Section 4. Section 5 is devoted to the presentation of a Branch and Bound method which uses the  $\epsilon$ -constraint and feasibility tests is presented in Section 6. In Section 7 we present computational testing.

## 2. Enumeration of the strict Pareto optima

To enumerate the strict Pareto optima a classic approach consists in iteratively solving  $\epsilon$ -constrained problems ([T'kindt and Billaut, 2002]).

The  $\epsilon$ -constrained problem considered in this paper is the following: we minimize the makespan subject to the additional constraint  $L_{max} \leq \epsilon$  where  $\epsilon$  is any given value. The corresponding scheduling problem is referred to as  $Q/r_i, d_i/\epsilon(C_{max}/L_{max})$ . It can be rewritten as:

$$(P_1) \quad Q/r_i, \tilde{d}'_i = d_i + \epsilon/C_{max}$$

Generally speaking, to each strict Pareto optimum  $s$  there exists an  $\epsilon$  value such that  $s$  is an optimal solution of the  $Q/r_i, d_i/\epsilon(C_{max}/L_{max})$  problem. The enumeration algorithm proceeds as follows. The value of  $\epsilon$  chosen at the first iteration is equal to a high value and let us denote by  $s_1$  the solution of the corresponding problem  $(P_1)$ . At the second iteration we set  $\epsilon = L_{max}(s_1) - 1$ . Again, problem  $(P_1)$  is solved and a new value  $\epsilon$  is set starting from the calculated solution. This process is iterated until problem  $(P_1)$  has no solution. To solve problem  $(P_1)$ , we can iteratively solve several problems:

$$(P_2) \quad Q/r_i, \tilde{d}'_i, C_{max} \leq \delta / -, \text{ where } \delta \text{ is any given value.}$$

Problem  $(P_2)$  is equivalent to:

$$(P_3) \quad Q/r_i, \tilde{d}_i = \min \{ \tilde{d}'_i, \delta \} / -$$

Let  $UB_{C_{max}}$  and  $LB_{C_{max}}$  be respectively an upper bound and a lower bound of the optimal  $C_{max}$  value for the  $Q/r_i, \tilde{d}'_i/C_{max}$  problem. The value of  $\delta$  used in problem  $(P_3)$  is defined by a dichotomic search, i.e. at each iteration  $\delta = (UB_{C_{max}} + LB_{C_{max}})/2$ . If the corresponding problem  $(P_3)$  is feasible, we set  $UB_{C_{max}} = \delta$ . Otherwise  $LB_{C_{max}} = \delta$ . This process stops when  $UB_{C_{max}} - LB_{C_{max}} = 1$  and we have  $C_{max}^* = UB_{C_{max}}$ .

We have transformed the solution of the optimization problem  $Q/r_i, d_i/C_{max}, L_{max}$  into the solution of a decision problem. We now focus on the solution of this decision problem.

### 3. A heuristic by progressive construction

This heuristic tries to solve the problem without studying all possible solutions. It can be seen as an alternate application of the Shortest Release Time (SRT) and Earliest Deadline (ED) rules. The SRT is applied, i.e. the job with the smallest value of  $r_i$  is scheduled, until the job with the smallest deadline must be scheduled otherwise, it would be infeasible if applying once more the SRT rule. A job is always scheduled on the machine on which it completes at the earlier. This algorithm requires

$o(n \log(n) + nm)$  time. Applying this heuristic enables to quickly detect if the problem is feasible, thus limiting the use of the exact method described in Section 6. If at the end of this heuristic no solution has been found, we must continue the solution of the feasibility problem by using an exact method. The algorithm of this heuristic is presented in Figure 1.

#### 4. Infeasibility tests

To study the infeasibility of the  $Q/r_i, \tilde{d}_i/-$  problem we can use two kinds of infeasibility tests which constitute sufficient conditions for deciding on the infeasibility of an instance, i.e. whenever they conclude to the infeasibility, the corresponding instance of the  $Q/r_i, \tilde{d}_i/-$  is infeasible. The first test, namely the preemptive approach, consists in enabling preemption of jobs. This relaxed decision problem can be polynomially solved by a network flow algorithm (see for instance [Brucker, 1995]). The second one, based on the energetic reasoning, has been first introduced in [Erschler et al, 1991] and [Lopez et al, 1992] and extended recently to uniform parallel machines in [T'kindt and Néron, 2004]. It is based on decomposing the scheduling horizon into time periods in which a mandatory part for each job, the energy requirement, has to be scheduled. If for a given time interval the sum of the energy requirements is greater than the processing capacities of the ressources then the problem is not feasible. The energy requirement of job  $J_i$  in the interval  $[t_1, t_2]$  when assigned to machine  $M_j$ , and denoted by  $W(i, j, t_1, t_2)$  is defined as follows:

$$W(i, j, t_1, t_2) = \begin{cases} \min\left(t_2 - t_1; \frac{p_i}{V_j}; \max\left(0, r_i + \frac{p_i}{V_j} - t_1; \max\left(0; t_2 - \tilde{d}_i + \frac{p_i}{V_j}\right)\right)\right) & \text{If } \frac{p_i}{V_j} \leq r_i - \tilde{d}_i \\ +\infty & \text{Otherwise} \end{cases}$$

Then we are faced, for each time interval, with the problem of assigning jobs to machines to test if all the mandatory parts fit in this interval. The details of this test is given in [T'kindt and Néron, 2004].

#### 5. Insertion of a job in a fixed sequence

This method enables us to add, in polynomial time, a job  $J_i$  to a partial solution  $\sigma_j$  on a given machine  $M_j$  or to decide that this insertion is not possible. It is used in the exact method provided in Section 6. Let us denote by  $t_k$  the starting time of job  $J_k$  in the partial schedule  $\sigma_j$ . We

```

/* $L_1$  is the unscheduled job with the smallest  $r_i^*$ */
/* $L_2$  is the unscheduled job with the smallest  $\tilde{d}_i^*$ */
While the list of unscheduled jobs is not empty
  If  $L_2$  can be scheduled on the machine with the smallest speed without infeasibility
  Then //Job  $L_2$  is not critical, so we try to assign the job  $L_1$ 
    We search for a machine on which job  $L_1$  completes without exceeding  $\tilde{d}_{L_1}$ 
    If a machine has been found
      Then the job is assigned on this machine and it is deleted from
      the set of the remaining jobs to schedule
    Else End: The heuristic has not found a solution
    EndIf
  Else //Job  $L_2$  is critical, we try to find a machine which accept it
    We search for the first machine which accept job  $L_2$ 
    If a machine is found
      Then the job is assigned to this machine and it is deleted from
      the set of the remaining jobs to schedule
    Else End: The heuristic has not found a solution
    EndIf
  EndIf
EndWhile
End: The heuristic has found a solution

```

Figure 1. A heuristic for the solution of the  $Q/r_i, \tilde{d}_i/-$  problem

first identify the different blocks of  $\sigma_j$  and the idle times between these blocks. A block is a set of jobs in which the end of each job coincides with the start of the next job, i.e. if  $J_k$  precedes immediately  $J_l$  then  $C_k = t_l$  (Figure 2). Next the method determines timeshifts that can be done on blocks in order to enable the assignement of job  $J_i$ .

To determine if a job  $J_i$  can be scheduled in the idle period before a block  $B_p$ , we calculate the maximum right timeshift of this block. The timeshift is always made to the right, so blocks that precede the block on which we calculate the maximum timeshift do not intervene in this calculation. The schedule  $\sigma_j$  is a schedule "left timeshifted", so the first job  $J_k$  of each block starts at the date  $r_k$ . In this section we present a mathematical formulation of the maximum timeshift of a block, and we give the scheme of the insertion method. We denote by  $b$  the number of blocks. Let  $B_p$  be the block number  $p$ ,  $\pi_p$  the duration of the idle time after block  $B_p$ , and  $\delta_p^{max}$  the maximum timeshift which the block can have without considering others blocks. We have  $\delta_p^{max} = \min_{k \in B_p} (\tilde{d}_k - C_k)$ . The maximum timeshift of block  $B_p$  taking account of the following blocks in  $\sigma_j$  is denoted by  $\delta_{max}$ .



**Property:** The maximum timeshift  $\delta max$  of a block  $B_p$  is defined by:

$$\delta max = \sum_{u=p}^{b-1} \pi_u + \min_{1 \leq u \leq b-1} \left( \delta_u^{max} - \sum_{v=u}^{b-1} \pi_v; \delta_b^{max} \right)$$

**Proof:**

We suppose, without loss of generality, that  $p = 1$ .

- For  $b=1$ , we have one block, so  $\delta max = \delta_1^{max}$ .
- For  $b=2$ , we distinguish two cases:
  - 1 If  $\delta_1^{max} \leq \pi_1$ , then  $\delta max = \delta_1^{max}$ .
  - 2 If  $\delta_1^{max} > \pi_1$ , then  $\delta max = \pi_1 + \min(\delta_1^{max} - \pi_1; \delta_2^{max})$ .

We note that, if  $\delta_1^{max} \leq \pi_1$ , the value  $\delta_1^{max} - \pi_1$  is negative. We have also:

$$\forall u, \delta_u^{max} \geq 0 \text{ and } \min(\delta_1^{max} - \pi_1; \delta_2^{max}) = \delta_1^{max} - \pi_1.$$

So, if  $\delta_1^{max} \leq \pi_1$ ,  $\delta max = \pi_1 + \delta_1^{max} - \pi_1 = \delta_1^{max}$ . Therefore, we can deduce that:

$$\boxed{\delta max = \pi_1 + \min(\delta_1^{max} - \pi_1; \delta_2^{max})}$$

- For  $b=3$ , we distinguish three cases:
  - 1 If  $\delta_1^{max} \leq \pi_1$ , then  $\delta max = \delta_1^{max}$ .
  - 2 If  $\delta_1^{max} > \pi_1$  and  $\min(\delta_1^{max} - \pi_1; \delta_2^{max}) \leq \pi_2$ , we are in the case where  $b = 2$ , i.e,  $\delta max = \pi_1 + \min(\delta_1^{max} - \pi_1; \delta_2^{max})$ .
  - 3 If  $\delta_1^{max} > \pi_1$  and  $\min(\delta_1^{max} - \pi_1; \delta_2^{max}) > \pi_2$ , the third block can decrease the timeshift of the first block. So, we also have the case  $b = 2$ , and
 
$$\delta max = \pi_1 + \pi_2 + \min(\min(\delta_1^{max} - \pi_1; \delta_2^{max}) - \pi_2; \delta_3^{max})$$

We can deduce the mathematical formulation:

$$\boxed{\delta max = \pi_1 + \pi_2 + \min(\delta_1^{max} - \pi_1 - \pi_2; \delta_2^{max} - \pi_2; \delta_3^{max})}$$

- By analogy we deduce the following mathematical formulation:

$$\boxed{\delta max = \sum_{u=1}^{b-1} \pi_u + \min_{1 \leq u \leq b-1} \left( \delta_u^{max} - \sum_{v=u}^{b-1} \pi_v, \delta_b^{max} \right)}$$

After having identified the blocks and calculated for each of them their maximum right timeshift, the insertion method starts trying to insert job  $J_i$ . First it tries to assign it before the first block. If this job cannot be inserted, we shift the first block to the right, and we try again to assign this job before the first block. Next if this job cannot be assigned, we shift the second block always to the right and we try to schedule  $J_i$  between the first and the second blocks, and so on for all blocks  $p$ ,  $\forall p = 1, \dots, b$ . If job  $J_i$  cannot be inserted between two blocks, we try to place it after the last block. Clearly, job  $J_i$  cannot be scheduled such that it starts before  $r_i$  or completes after  $\tilde{d}_i$ . This method requires  $o(n)$  time.

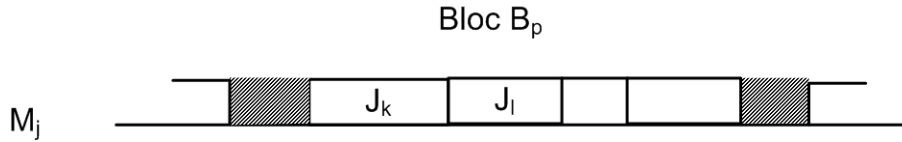


Figure 2. A block of the partial schedule  $\sigma_j$

## 6. A Branch and Bound procedure

The branch and bound method is widely used to solve scheduling problems. It is a solution procedure which systematically examines all possible combinations of sequencing jobs on machines. Before running the branch and bound method, the heuristic described in Section 3 is applied to try to decide on the feasibility of the problem. If it fails then the two infeasibility tests given in Section 4 are used to detect whether the instance is infeasible or not. If both feasibility and infeasibility have not been heuristically stated the branch and bound algorithm is run. This method calculates a feasible schedule, if it exists, by exploring a search tree in which each node is defined by, when applied to our problem, a partial schedule and the set of the remaining jobs to schedule. Starting from the root node, child nodes are created by assigning a job on the different machines. From the previous nodes, the method continues by selecting at each level a job and assign it on all machines. We continue in this way to create child nodes until we arrive to the leaf nodes. Let us consider a node for which we search to assign job  $J_i$  on machine  $M_j$ . We denote by  $\sigma_j$  the set of jobs assigned on the machine  $M_j$ , and  $\Omega$  the set of unscheduled jobs. We first heuristically try to detect if the job can be scheduled on the machine  $M_j$  in two steps. The first one consists

in inserting  $J_i$  in the sequence  $\sigma_j$  without introducing infeasibility (cf. Section 5), which can be achieved in  $o(n)$  time. If this insertion is not possible then the heuristic of Section 3 is applied on  $\sigma_j \cup \{J_i\}$  and to machine  $M_j$ . During one of these two steps, if we have constructed a feasible partial schedule then the current node is conserved. Otherwise we try to detect the infeasibility of this node by using the energetic test introduced in Section 4: if an assignment of a job on a machine is detected infeasible, the corresponding node is pruned. If the current node cannot be pruned by the infeasibility test, we have to exactly solve the  $1/r_i, \tilde{d}_i/-$  problem related to machine  $M_j$  with the branch and bound procedure of Carlier ([Carlier, 1982]). It is possible to get a solution from this algorithm, if this algorithm detects that the problem is feasible. This solution is kept as the new partial schedule of the child node. The current node is pruned if job  $J_i$  cannot be assigned to machine  $M_j$ .

## 7. Computational testings

Several computational experiences have been conducted in order to evaluate the performance of the heuristic by progressive construction and the exact method. Data sets which are used in the computational experiments are generated as follows: we generate problems with  $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$  and  $m \in \{2, 4, 6\}$ . To rule the distribution of due dates and release dates, two factors, namely  $L$  and  $R$  are used. Factor  $L$  is used to center the distribution of due dates and release dates while factor  $R$  rules their spreading. For each problem size  $(n; m)$ , different values of factor  $R$  are tested:  $R \in \{0.2; 0.4; 0.6; 0.8; 1.0; 1.2; 1.4; 1.6\}$ . The value of factor  $L$  in the uniform parallel machine problem is defined by  $L = \frac{\bar{P}}{V_{min}}$  with  $V_{min} = \min_{j=1, \dots, m} V_j$ , and  $\bar{P} = \sum_{i=1}^n p_i$ . The machine speeds are drawn at random between 1 and 10 using an uniform distribution law. Absolute job processing times  $p_i$  are drawn at random between 10 and 100 using an uniform distribution law. For each job  $J_i$ , its due date  $d_i$  is drawn at random between  $(L - \frac{R \times \bar{P}}{2V})$  and  $(L + \frac{R \times \bar{P}}{2V})$ , with  $\bar{V} = \sum_{j=1}^m V_j$  using an uniform distribution law. Similarly, the release dates  $r_i$  are generated between  $[d_i - 3\frac{p_i}{V_{max}}; d_i - \frac{p_i}{V_{max}}]$ , with  $V_{max} = \max_{j=1, \dots, m} V_j$ . In case where  $r_i < 0$ , we set  $d_i = d_i - r_i$  and  $r_i = 0$ . For each tuple  $(n; m; R)$ , 30 instances are generated thus leading to the total of 240 instances per problem size. The experimental results show that the branch and bound algorithm is capable of solving problems with up to 30 jobs and 6 machines in a reasonable amount of time. More details about the computational results will be provided at the conference.

These results will also illustrate how it is hard to conduct the enumeration of strict Pareto optima.

## References

- Brucker P., Scheduling algorithms, *Springer, Heidelberg*, (1995).
- Lahrichi A., Ordonnancements: la notion de parties obligatoires et son application aux problèmes cumulatifs, *R.A.I.R.O.-R.O*,26:241–262, (1982).
- Carlier J., The one-machine sequencing problem, *European Journal of Operational Research*, 11:42–47, (1982).
- Erschler J., Lopez P., and Thuriot C., Raisonnement temporel sous contraintes de ressources et problèmes d’ordonnancement, *Revue d’Intelligence Artificielle*,5:7–32, (1991).
- Garey M.R. and Johnson D.S. Computers and intractability: a guide to the theory of NP-completeness. *W.H. Freeman and Company*, (1979).
- Graham R.L., Lawler E.L., Lenstra J.K., and Rinnooy Kan A.H.G. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*,5:287–326, (1979).
- Lopez P., Erschler J. and Esquirol P., Ordonnancements de tâches sous contraintes: Une approche énergétique, *R.A.I.R.O.- A.P.I.I*,26:453–481, (1992).
- T’kindt V. and Billaut J-C. Multicriteria Scheduling: Theory, Models and Algorithms. *Springer (Heidelberg)*, (2002).
- T’kindt V. and Néron E., Energetic and preemptive feasibility tests for parallel machines scheduling, *Research report, Laboratoire Informatique, Université François-Rabelais de Tours*, 1–15, (2004).
- Tercinet F., Méthodes arborescentes pour la résolution des problèmes d’ordonnancement, conception d’un outil d’aide au développement, *Ph.D.Thesis, Laboratoire Informatique, Université François-Rabelais de Tours*, (2004).

# HEURISTIC APPROACHES TO A COMMON DUE DATE EARLINESS-TARDINESS SCHEDULING PROBLEM

Valery S. Gordon, Alexander A. Tarasevich

National Academy of Sciences of Belarus, United Institute of Informatics Problems, Minsk, Belarus, {gordon@newman.bas-net.by, Tarasevich.Alexander@gmail.com}

**Abstract:** In this paper we compare simulated annealing and tabu search approaches to a single machine common due date assignment and scheduling problem with jobs available at different times. The objective is to minimize the total weighted sum of earliness, tardiness and due date costs.

## 1. INTRODUCTION

Among the scheduling problems considered within a “Just-in-Time” (JIT) concept there are problems where jobs are to be completed as close as possible to a common due date. As an example of such problems consider the following problem arising in food industry. Fruits or vegetables from different farms are to be gathered and processed at a plant. If they are delivered too early it is necessary to pay for their storage. Otherwise, if they are processed too late, there would be a loss of money due to a large number of spoiled fruits or vegetables. The situation can be modeled by a following single machine scheduling problem of minimizing earliness-tardiness costs with different ready times of jobs and assignable common due date.

Consider a scheduling situation when  $n$  jobs are to be processed by a single machine under common due date  $d$ , which is to be assigned. Each job  $j$ ,  $j = 1, 2, \dots, n$ , becomes available at moment  $r_j$  (*release date, ready time*) and has *processing time*  $p_j$ . Let  $S_j$  and  $C_j$  be *starting* and *completion times* of job  $j$  in a certain schedule  $s$ . Let  $E_j = \max\{0, d - C_j\}$  and  $T_j = \max\{0, C_j - d\}$  be *earliness* and *tardiness*

of job  $j$ . The objective is to minimize  $f(d, s) = \sum_{j=1}^n (\alpha E_j + \beta T_j + \gamma d)$ , where  $\alpha > 0$ ,  $\beta > 0$  and  $\gamma > 0$

are earliness, tardiness and due date per unit penalties. So, we have to find an optimal schedule  $s^*$  and an optimal due date  $d^*$  to minimize total penalty  $f(d, s)$ . Note that permitting  $\beta = 0$  leads to  $d^* = 0$  and to optimality of any feasible schedule. The problem cases with  $\alpha = 0$  or  $\beta \leq \gamma$  are equivalent to the strongly NP-hard problem of minimizing the sum of completion times subject to given release dates [11], which is well studied. Further we consider the case  $\beta > \gamma$ . Extending the standard scheme for scheduling notation [10], we refer to our due date assignment problem as  $1|r_j, d_j := d | \sum (\alpha E_j + \beta T_j + \gamma d)$ .

Panwalkar, Smith and Seidmann [12] have considered the case of this problem when all jobs are available at time zero. They show that the optimal common due date coincides with the completion time of the  $[K]$ -th job, and provide a polynomial time algorithm for solving the problem. Here  $[j]$  denotes a job in position  $j$  of job sequence and  $K$  is the smallest integer greater or equal to  $n(\beta - \gamma)/(\alpha + \beta)$ .

Cheng, Chen and Shakhlevich [3] consider the general case of the problem (with non-zero ready times) and proposed a heuristic algorithm. This heuristic consists of two parts. First, determine a sequence  $\sigma^*$  optimal for the relaxed problem with zero ready times by Panwalkar et al. algorithm. Then obtain a schedule  $s_h^*$  and a due date  $d_h^*$  for the predetermined sequence  $\sigma^*$  of jobs. We refer to this algorithm as to CCS heuristic. The worst case analysis of the CCS algorithm gives the following result [3]:  $f(s_h^*, d_h^*) / f(s^*, d^*) \leq 1 + R$ . Here  $s^*$  and  $d^*$  are the optimal schedule and the optimal due date,  $R = \max_j \{r_j\}$ .

Application of meta-heuristics, namely evolutionary strategies, simulated annealing (SA) and threshold accepting, to a problem with zero ready times, but with  $\alpha = \alpha(j)$  and  $\beta = \beta(j)$  is considered in [5].

We compare SA and tabu search (TS) approaches to the general case of the problem with constant  $\alpha$  and  $\beta$ . Both approaches use the following polynomial-time algorithm [3] proposed for the case of the problem when the sequence of job processing is predetermined.

ALGORITHM 1.

1. Determine starting times of the jobs as follows:  $S_{[1]}(\sigma) = r_{[1]}$ ,  $S_{[i]}(\sigma) = \max\{r_{[i]}, C_{[i-1]}(\sigma)\}$ .
2. Calculate  $u := \lceil n(\beta - \gamma) / \beta \rceil$ .
3. Move jobs in positions  $1, 2, \dots, u-1$  to job  $u$  so that there will be no idle intervals between them.
4. Calculate  $K := \lceil n(\beta - \gamma) / (\alpha + \beta) \rceil$  and set  $d_h^* := C_{[K]}$ , where  $[K]$  is a job in the  $K$ -th position of the sequence  $\sigma$ .

The paper is organized as follows. In section 2 we propose a SA approach to the problem and describe corresponding heuristic algorithm. Section 3 contains a TS algorithm description. The computational results on comparison of SA and TS algorithms are presented in section 4.

## 2. SA HEURISTIC

Local search optimization techniques solve problems by moving from one (current) solution to another (neighboring). The efficiency depends on deriving neighbors based on the previous solution and on the rule of neighbor acceptance after comparing optimality criteria. Usual local search methods have great chance to get trapped in a local minimum, as most of them accept only better neighbors.

SA approach is widely used for the local search in optimization problems due to its ability to overcome local minimums. The name “*simulated annealing*” derives from the analogy with the physical process of annealing: obtaining highly-structured low-energy states of solid in a heat bath through lowering temperature gradually. In scheduling problems, a schedule corresponds to the solid and a penalty function corresponds to its energy. For detailed description of SA, see [1,4,9].

We use the especial ability of SA to accept, with some decreasing probability  $e^{-\frac{(f^* - f')}{t}}$ , neighboring solutions which do not improve current solution. Here  $f^*$  is a value of the objective function for the current solution and  $f'$  is the corresponding value for the neighboring solution;  $t$  is a control parameter. This allows SA heuristic to overcome local minimums in most cases.

An idea of the proposed approach is to apply SA to a schedule obtained by algorithm 1 for some initial sequence  $\sigma$  of jobs. Preliminary computational experiments show that applying algorithm 1 to a random

sequence or to a sequence obtained by Panwalkar et al. algorithm gives worse results than applying it to a sequence of jobs ordered by their release dates. So we use the latter sequence as an initial sequence  $\sigma$ .

For our scheduling problem, initial value of  $t = t_0$  was chosen from the equality 
$$e^{-\frac{(f_{\min}^* - f'_{\max})}{t_0}} = 0.999$$
, where  $f_{\min}^*$  и  $f'_{\max}$  are the best and the worst values of optimality criteria obtained by CCS algorithm from the set of 10000 random problem instances. This condition gives a great chance at early stages to accept neighbors which do not improve current solution. After transition (attempt of moving) from current to neighboring solutions,  $t$  is reduced:  $t := Rt$ . In literature,  $R$  usually lies between 0.8 and 0.99. After a number of preliminary computational experiments we obtain that  $R = 0.85$  gives us the best solutions and the best performance among 0.8, 0.85, 0.9, 0.95, 0.99. So we use  $R = 0.85$  in our heuristic.

Here are the main steps of the proposed approach.

Construct starting sequence  $\sigma$  of jobs by arranging them in non-decreasing order of their release dates. Take solution  $(s^o, d^o, f^o)$  as the best solution where  $f^o$  is a value of penalty function for the schedule  $s^o$  obtained by applying algorithm 1 for  $\sigma$  and  $d^o$  is a due date obtained by algorithm 1.

Take the best solution as the current one. Let  $f^*$  be a value of the penalty function for the current solution. Repeat the following transition  $n$  times: Obtain sequence  $\sigma'$  for neighboring solution by interchanging two jobs randomly in the sequence of current solution. Apply algorithm 1 to  $\sigma'$  to get the neighboring solution with the value  $f'$  of penalty function. If the neighboring solution is not worse than the current one, i.e.  $f' \leq f^*$ , then take it as a current. If the current solution is better than the best one, i.e.  $f^* < f^o$ , then take current solution as the best. If neighboring solution is worse than the current one, i.e.  $f' > f^*$ , then accept it as a current with probability  $\text{EXP}((f^* - f')/t)$ . Reduce control parameter  $t := Rt$ .

If cycle 2 repeats  $10n$  times or if the best solution was not improved during last 10 cycles then stop: the best solution is the result of heuristic. Else repeat cycle 2 again.

There is a difference in such approach from the usual SA method. Algorithm returns to the best solution after  $n$  transitions and waits 10 cycles without the best solution improvement until stop. We choose the limit  $10n$  times of repeating cycle 2 since preliminary experiments show that further repeating does not lead to considerable improvement of the results. These limitations allow us to reduce run time of the algorithm considerably.

In the worst case it takes  $O(n^3)$  units of time to run the algorithm. Detailed description of the algorithm is given below.

#### ALGORITHM 2.

1. Set  $t := t_0$ . Construct sequence  $\sigma = ([1], [2], \dots, [n])$  by arranging jobs in non-decreasing order of their release dates and determine a schedule  $s^*(\sigma)$  and a due date  $d^*(\sigma)$  using algorithm 1.
2. Calculate penalty function for the schedule  $s^*(\sigma)$  and the due date  $d^*(\sigma)$ :  

$$f^* = f(d^*, s^*).$$
3. Let schedule  $s^*(\sigma)$ , due date  $d^*(\sigma)$  and penalty function value  $f^*$  be current solution, and let the best solution  $(s^o, d^o, f^o) := \text{current solution } (s^*, d^*, f^*)$ .
4.  $\text{CyclesWOImpr} := 0$ ;  $\text{Cycles\_counter} := 0$ .
5.  $i := 0$ ;  $\text{Cycles\_counter} := \text{Cycles\_counter} + 1$ .
6.  $i := i + 1$ .
7. Exchange jobs in position  $i$  and randomly chosen position  $j$  in the sequence  $\sigma$  and obtain neighboring solution  $(s^*(\sigma'), d^*(\sigma'), f')$  for new sequence  $\sigma'$ :

8. Determine  $d^*(\sigma')$  and  $s^*(\sigma')$  using algorithm 1 for sequence  $\sigma'$ . Calculate  $f' = f(d^*(\sigma'), s^*(\sigma'))$ .
9. Compare the neighboring solution with the current:
  - 9.1 **if**  $f' \leq f^*$  **then** current solution := the neighboring solution.
  - 9.2 **if**  $f' < f^o$  **then** the best solution := the neighboring solution;  $CyclesWOImpr := 0$ .
  - 9.3 **if**  $f' > f^*$  **then if**  $RANDOM[1] < EXP((f^* - f')/t)$  **then** current solution := the neighboring solution.
10. Reduce control parameter  $t := Rt$ .
11. **if**  $i < n$  **then goto** step 6.
12. Current solution := the best solution;  $CyclesWOImpr := CyclesWOImpr + 1$ ;
13. Check stop conditions:
  - 13.1 **if**  $CyclesWOImpr = 10$  **then goto** step 15.
  - 13.2 **if**  $Cycles\_counter = n^2$  **then goto** step 15.
14. **goto** step 5.
15. Current solution is the best solution obtained by heuristic. **stop**.

Notations:

$Cycles\_counter$  variable counts number of iteration cycles (steps 6-9) with the same  $t$  value.

$CyclesWOImpr$  variable counts number of cycles passed without improving the best (not current) solution.

$RANDOM[1]$  is the function that returns random real number from the range  $[0,1]$ .

### 3. TABU SEARCH

Tabu search (TS) is a local search method which uses special tabu list of visited solutions to prevent searching in the previously explored areas. Together with the best solution, TS keeps information on the itinerary through the last solutions visited [6,7,8]. Unlike the greedy algorithm of local search, in TS the visited but not accepted solutions are placed into the tabu list, and algorithm passes by solutions that are in the tabu list. When the list is full, some solution (the best or the first in tabu list) is removed from the list to accept the last visited solution. The efficiency of such approach depends on a structure of the set of the problem possible solutions. The length of tabu list and its structure affect the speed of problem solving and the quality of the obtained results.

For our problem, the neighbor solution is obtained by exchanging randomly chosen pairs of jobs. After preliminary experiments the length of tabu list is set to  $n\sqrt{n}$  and the parameter for defining the size of neighborhood is chosen to be  $n/5$ . To spread the exploration effort over different region of the set of solutions, we choose parameter  $\sqrt{n}$  to get a diversified sequence by exchanging job pairs. The number of the diversifications is chosen to be  $n/5$ .

In the worst case it takes  $O(n^4)$  units of time to run the algorithm. Detailed description of the algorithm is given below

#### ALGORITHM 3

- 1 Construct sequence  $\sigma = ([1],[2],\dots,[n])$  by arranging jobs in non-decreasing order of their release dates. Determine a schedule  $s^*(\sigma)$  and a due date  $d^*(\sigma)$  using algorithm 1 and calculate  $f^*(s^*(\sigma), d^*(\sigma))$ . Let the best solution  $(s^o, d^o, f^o) := \text{current solution } (s^*, d^*, f^*)$ .
- 2  $Restarts\_number = 0$ .



- 3  $i = 1$ .
- 4  $j = 0$ . Let the best solution in the neighborhood  $(s'', d'', f'') := \text{current solution } (s^*, d^*, f^*)$ .
- 5  $j = j + 1$ . Exchange job  $i$  with randomly chosen job in the sequence of the current solution to obtain new sequence  $\sigma'$  and the corresponding solution  $(s(\sigma'), d(\sigma'), f')$  by algorithm 1.
- 6 **if** obtained solution  $(s(\sigma'), d(\sigma'), f')$  exists in the Tabu list **then** move it to the end of the list and **then goto step 5**.
- 7 **if**  $f' < f''$  **then** the best solution in the neighborhood  $(s'', d'', f'') := \text{obtained solution } (s(\sigma'), d(\sigma'), f')$ ; **else** put obtained solution into the Tabu list.
- 8 **if** list length  $> n\sqrt{n}$  **then** remove the first solution from the list.
- 9 **if**  $j < n/5$  **then goto step 5**.
- 10 **if**  $f'' < f^*$  **then** current solution  $(s^*, d^*, f^*) := \text{the best in the neighborhood solution } (s'', d'', f'')$ ; **else** put solution  $(s'', d'', f'')$  into the Tabu list.
- 11  $i = i + 1$ ; **if**  $i \leq n$  **then goto step 4**.
- 12 **if**  $f^* < f^o$  **then** the best solution  $(s^o, d^o, f^o) := \text{current solution } (s^*, d^*, f^*)$ .
- 13 Get new (deversified) sequence by exchanging randomly chosen  $\sqrt{n}$  job pairs.
- 14  $\text{Restarts\_number} = \text{Restarts\_number} + 1$ . **if**  $\text{Restarts\_number} < n/5$  **then goto step 3**.
- 15 The result is the best solution  $(s^o, d^o, f^o)$ . **stop**.

Notations:

$i, j$  are the cycles counters.

$\text{Restarts\_number}$  is the number of diversifications.

## 4. COMPUTATIONAL EXPERIMENT

Algorithms 1, 2, 3 were programmed in Visual C++ 6.0 and tested on computer with 1800 MHz Athlon XP processor.

Problem instances were created randomly using benchmarks for single machine scheduling problems by Biskup and Feldmann [2]. There are four control parameters for generating problem instances:

$n$  is the number of jobs ( $n = 20, 50, 100, 1000$ ).

$p_{\max}$  is the maximal job processing time ( $p_{\max} = 20, 50, 100$ ).

$r_{\max} = knp_{\max}$  is the maximal release date ( $k = 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.5, 1, 1.5, 2, 1/n$ ).

$\mu_{\max}$  is the maximum for  $\alpha, \beta, \gamma$  values ( $\mu_{\max} = 10, 20, 30$ ).

For each problem instance,  $\alpha, \beta, \gamma$  are chosen randomly from the range  $[1, \mu_{\max}]$  taking into account  $\beta > \gamma$  condition;  $p_j$  and  $r_j$  are chosen from the ranges  $[1, p_{\max}]$  and  $[1, r_{\max}]$ , respectively. Preliminary experiments show that the run time of the algorithms depend on  $r_{\max}$  but not on  $p_{\max}$  and  $\mu_{\max}$ . For problem instances with  $n = 1000$ , smaller values of  $p_{\max}$  were taken to avoid memory overflow :  $p_{\max} = 6, 16, 33$ .

To compare SA, TS and CCS heuristics we calculate  $f_{sa}/f_{ccs}$ ,  $f_{ts}/f_{ccs}$  and  $f_{ts}/f_{sa}$ , where  $f_{sa}$ ,  $f_{ts}$  and  $f_{ccs}$  are the penalty function values obtained by the SA, TS and CCS algorithms for the same problem instances.

It is natural that the proposed heuristics obtain better solution than those by CCS algorithm for the same problem instances. For all combinations of  $n$  and  $k$ , we have  $f_{sa}/f_{ccs} < 1$  and  $f_{ts}/f_{ccs} < 1$  for all problem instances. The range of average ratios  $f_{sa}/f_{ccs}$  is from 0.535 to 0.966; the  $f_{ts}/f_{ccs}$  ratios lay in the range from 0.495 to 0.966. Note that for small  $k$ , results obtained with SA and TS heuristics are closer to those obtained with CCS heuristic. This can be explained by the nature of CCS algorithm: using a sequence for the relaxed problem with zero ready times, it obtains solutions close to optimal when release dates  $r_j$  are distributed in a small range.

The average value of  $f_{ts}/f_{sa}$  is close to 1 (Table 1), though for small  $n$  values SA approach gives slightly better solutions than those by TS heuristic.

Tables 2 and 3 show the run time of SA and TS algorithms for  $n = 20, 50, 100, 1000$  depending on release dates (parameter  $k$ ). Average amount of time necessary for solving the same problem instance is smaller for TS algorithm.

For large values of  $n$  ( $n = 1000$ ) TS algorithm obtains better solutions though it takes much more run time to solve a problem comparing to SA heuristic.

## 5. SUMMARY

We consider single machine common due date assignment and scheduling problem under given release dates for the jobs. The problem is known to be *NP*-hard. An  $O(n \log n)$  heuristic algorithm was proposed for the problem by Cheg, Chen and Shakhlevich [3] (CCS algorithm). We compare simulated annealing (SA) and tabu search (TS) approaches to the problem. Preliminary computational experiments led to choosing a sequence of jobs ordered by their release dates as an initial solution in both approaches. In deriving neighbors we also took into account release dates of the jobs. The number of iterations was restricted to get  $O(n^3)$  running time for SA algorithm and  $O(n^4)$  time for TS algorithm.

Computational experiments show that both approaches can be used for the problem under consideration giving better results than CCS algorithm, and for smaller number  $n$  of jobs ( $n \leq 100$ ) SA algorithm gives better solutions yielding to TS algorithm in time. For larger number of jobs the results are opposite: TS algorithm leads to better solutions taking more time. Further, a comparison of the performance with other meta-heuristic approach based on genetic algorithm can be considered.

## ACKNOWLEDGEMENTS

The research was partially supported by INTAS project 03-51-5501.

## REFERENCES

1. Aarts, E.H.L., Korst, J.H.M., van Laarhoven P.J.M.: Simulated Annealing. In: Aarts, E.H.L., Lesntra, J.K. (Eds.), *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997, 91-120.
2. Biskup, D., Feldmann, M.: Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research* 28 (2001), 787-801.
3. Cheng, T.C.E., Chen, Z-L., Shakhlevich, N.V.: Common due date assignment and scheduling with ready times. *Computers & Operations Research* 29 (2002), 1957-1967.
4. Dowsland, K.A. Simulated Annealing. In: Reeves C.R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. Blackwall, Oxford (1993), 20-69.

5. Feldmann, M., Biskup, D.: Single machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering* 44 (2003), 307-323.
6. Glover, F.: Tabu search: part I. *ORSA Journal of computing* 1 (1989), 190-206.
7. Glover, F.: Tabu search: part II. *ORSA Journal of computing* 2 (1990), 4-32.
8. Hertz, A., Taillard, E., de Werra, D.: Tabu Search. In: Aarts, E.H.L., Lenstra, J.K. (Eds.), *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997, 121-136.
9. Kirkpatrick, C.D. Gelatt, Jr., Vecchi, M.P.: Optimization by simulated annealing. *Science* 220 (1983), 671-680.
10. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: Algorithms and complexity. In: Graves, S.C., Zipkin, P.H., Rinnooy Kan, A.H.G. (Eds.), *Logistics of Production and Inventory*. Handbooks in Operations Research and Management Science, vol. 4, North-Holland, Amsterdam, 1993, 445– 522.
11. Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1 (1977), 343–62.
12. Panwalkar, S.S., Smith, M.L., Seidmann, A.: Common due date assignment to minimize total penalty for the one machine scheduling problem. *Operations Research* 30 (1982), 391–399.

## TABLES

Table 1.  $f_{ts} / f_{sa}$  value for each  $n$  depending on  $k$ .

| $K$   | $n=20$ | $n=50$ | $n=100$ | $n=1000$ |
|-------|--------|--------|---------|----------|
| $1/n$ | 1.018  | 1.003  | 1.001   | 1        |
| 0.01  | 1.021  | 1.002  | 1.001   | 1        |
| 0.03  | 1.016  | 1.003  | 1.001   | 0.998    |
| 0.05  | 1.016  | 1.002  | 1       | 0.996    |
| 0.1   | 1.013  | 1.003  | 1       | 0.992    |
| 0.15  | 1.015  | 1.005  | 0.999   | 0.982    |
| 0.2   | 1.014  | 1.009  | 1       | 0.985    |
| 0.25  | 1.029  | 1.011  | 1.001   | 1.104    |
| 0.3   | 1.024  | 1.016  | 1.005   | 0.997    |
| 0.5   | 1.042  | 1.022  | 1.017   | 0.925    |
| 0.7   | 1.013  | 1.012  | 1.007   | 1        |
| 1.0   | 1.010  | 1.005  | 1.003   | 0.986    |
| 1.5   | 1.002  | 1.003  | 1       | 0.962    |
| 2.0   | 1.017  | 1.001  | 1       | 0.77     |

Table 2. Run time (in seconds) of SA algorithm to solve the problem for each  $n$  depending on  $k$ .

| $K$   | $n=20$ | $n=50$ | $n=100$ | $s$ |
|-------|--------|--------|---------|-----|
| $1/n$ | 0.001  | 0.014  | 0.151   | 260 |
| 0.01  | 0.001  | 0.014  | 0.152   | 259 |
| 0.03  | 0.001  | 0.013  | 0.151   | 258 |
| 0.05  | 0.001  | 0.012  | 0.148   | 260 |
| 0.1   | 0.001  | 0.011  | 0.140   | 259 |
| 0.15  | 0.001  | 0.009  | 0.128   | 259 |
| 0.2   | 0.001  | 0.008  | 0.094   | 260 |
| 0.25  | 0.001  | 0.007  | 0.068   | 259 |
| 0.3   | 0.001  | 0.007  | 0.056   | 258 |
| 0.5   | 0.001  | 0.005  | 0.031   | 113 |
| 0.7   | 0.001  | 0.004  | 0.024   | 151 |
| 1.0   | 0.001  | 0.004  | 0.027   | 174 |
| 1.5   | 0.001  | 0.004  | 0.029   | 194 |
| 2.0   | 0.001  | 0.004  | 0.034   | 190 |

Table 3. Run time (in seconds) of TS algorithm to solve the problem for each  $n$  depending on  $k$ .

| $K$   | $n=20$ | $n=50$ | $n=100$ | $n=1000$ |
|-------|--------|--------|---------|----------|
| $1/n$ | 0      | 0.003  | 0.059   | 2292     |
| 0.01  | 0      | 0.003  | 0.061   | 2846     |
| 0.03  | 0      | 0.003  | 0.042   | 3646     |
| 0.05  | 0      | 0.003  | 0.040   | 3900     |
| 0.1   | 0      | 0.002  | 0.028   | 4661     |
| 0.15  | 0      | 0.002  | 0.027   | 5709     |
| 0.2   | 0      | 0.002  | 0.020   | 11837    |
| 0.25  | 0      | 0.002  | 0.018   | 6968     |
| 0.3   | 0      | 0.002  | 0.017   | 6834     |
| 0.5   | 0      | 0.001  | 0.009   | 5467     |
| 0.7   | 0      | 0.001  | 0.007   | 3887     |
| 1.0   | 0      | 0.001  | 0.007   | 3304     |
| 1.5   | 0      | 0.001  | 0.009   | 3324     |
| 2.0   | 0      | 0.001  | 0.009   | 2146     |

# THE SINGLE MACHINE JUST-IN-TIME SCHEDULING PROBLEM WITH PREEMPTIONS

Yann Hendel

*LIP6*

*8 rue du capitaine Scott*

*75015 Paris, France*

Yann.Hendel@lip6.fr

Francis Sourd

*LIP6*

*8 rue du capitaine Scott*

*75015 Paris, France*

Francis.Sourd@lip6.fr

**Abstract** This paper deals with a variation of the single machine earliness/tardiness scheduling problem: preemption is allowed and, in order to penalize job interruptions, earliness costs depend on the starting time of the job while tardiness costs depend on the end times as in the usual models. We propose dominance properties and a polynomial time algorithm is given to solve the particular case where the sequence of the start and completion of the jobs is fixed.

**Keywords:** Scheduling, Single machine, just-in-time, preemption

## Introduction

Just-in-time scheduling has interested both practitioners and researchers for over a decade. A very common idea is to recognize that a job that completes either tardily or early in a schedule induces extra costs. Most of the time, just-in-time costs are function of the job completion time. However, in a context where preemption is allowed, such functions may not render the will of the scheduler: indeed, when starting

a job, the goal is generally to complete it as soon as possible so that it can be removed from the production line. If the cost only depends on the completion time of the job, idle time within the execution of the job will not be penalized. In order to model these costs, it seems natural to attach the earliness costs to the start time of the job, while tardiness costs are still tied to the completion of the jobs. Such an approach has been used in [Hendel and Sourd, 2003], for the job-shop problem with two jobs. In this problem, preemption is not allowed but the idea is to penalize waiting times in between operations.

In this paper, we adapt the classic one-machine earliness/tardiness problem when preemption is allowed and to render the earliness/tardiness costs, we will consider that each job has two due dates instead of one: one tied to the start time of the job and the other tied to the completion time of the job.

Formally, we consider a set of jobs  $\mathcal{J} = \{J_1, \dots, J_n\}$  which has to be scheduled on a single machine. Each job  $J_j \in \mathcal{J}$  has a processing time  $p_j$ . We define tardiness costs as  $T_j = \max(0, C_j - d_j^c)$  where  $C_j$  is the completion time of the job and  $d_j^c$  is the *ideal completion time* of job  $J_j$ , which corresponds to the usual due date. Earliness costs are defined as  $E_j = \max(0, d_j^s - S_j)$  where  $S_j$  is the starting time of job  $J_j$  and  $d_j^s = d_j^c - p_j$  is the *ideal starting time* of job  $J_j$ . Therefore,  $J_j$  would ideally be processed between  $d_j^s$  and  $d_j^c$ , which would cause no penalty costs. We want to minimize the total cost  $\sum_{j=1}^n (\alpha_j \cdot E_j + \beta_j \cdot T_j)$ .

We first notice that if preemption is not allowed, the problem is equivalent to the classic earliness/tardiness problem and if preemption is allowed but earliness costs are tied to the completion of the jobs, then the problem is equivalent to the single machine problem with only weighted tardiness costs.

The  $1||\sum_i f_i(C_i)$  problem has been extensively studied. One way to approach this NP-complete problem [Garey et al., 1988], is to solve a particular case where the job order is given i.e. when the job sequence is fixed; then eventually use it in a branch and bound or in a metaheuristic. [Garey et al., 1988] propose for this subproblem a direct algorithm based on the blocks of adjacent jobs in  $O(n \log n)$ , valid for the just-in-time problem with symmetric earliness and tardiness penalties. [Chrétienne and Sourd, 2003] present a generalization for convex functions. Finally, [Sourd, 2002] proposes a dynamic programming algorithm for general non-convex piecewise linear cost functions (which also considers eventual costs for the idle periods) whose complexity depends on the number of segments of the cost functions given in input.

In this paper, we will use the same approach. However, in contrast to the non-preemptive problem, the notion of job sequencing is not clear in

the preemptive case. Indeed, in an optimal solutions, starting time order usually differs from end time order. Let us first assume that the order of the starting times is given (but the order of the completion times is still to be determined). It can be easily shown that this problem is NP-complete by reducing the weighted tardiness problem, which is NP-complete [Lenstra et al., 1977], to it.

Therefore, we will study the particular case where the order of the starting and completion times of the jobs is given, that is we assume that all the events  $S_1, C_1, S_2, C_2, \dots, S_n, C_n$  are completely ordered. Basically, this constraint is formulated as a list of  $2n - 1$  inequalities (for instance  $S_1 \leq S_2 \leq C_2 \leq C_1 \leq S_3 \leq C_3$ ). Clearly, this ordering must satisfy  $S_i \leq C_i$  for each  $i$ .

In section 1, we propose dominance properties for this problem, and extract a dominant set of sequences. In section 2, we provide a strongly polynomial algorithm for solving the problem with dominant sequences.

## 1. Dominance properties

For any given sequence (that satisfies  $S_i \leq C_i$  for each  $i$ ), we can obtain an optimal schedule by solving the following linear program:

$$\begin{aligned}
 \text{(LP) : } \min \quad & \sum_{i=1}^n (\alpha_i \cdot E_i + \beta_i \cdot T_i) \\
 \text{s.t.:} \quad & E_i \geq d_i^s - S_i, & i = 1 \dots n \\
 & T_i \geq C_i - d_i^c, & i = 1 \dots n \\
 & C_j - S_i \geq p_{ij}, & ((i, j) | C_j \geq S_i) \\
 & S_i, C_i \geq 0 & i = 1 \dots n \\
 & E_i, T_i \geq 0 & i = 1 \dots n
 \end{aligned}$$

The constants  $p_{ij}$  describe the mandatory processing time that occurs between  $S_i$  and  $C_j$ , which is the processing of all the jobs that have started after  $S_i$  and finished before  $C_j$ . This LP have  $O(n)$  variables but  $O(n^2)$  constraints. In the following, we will show that we can obtain by a direct strongly polynomial algorithm an optimal schedule for a set of restricted sequences with a better theoretical complexity than the one of the LP.

**Property 1.** *A sequence that contains two jobs such that  $S_i \leq S_j \leq C_i \leq C_j$  is dominated.*

*Proof:* We consider the time intervals when  $J_i$  and  $J_j$  are executed, the sum of the length of these intervals being  $p_i + p_j$ , the execution intervals of  $J_i$  and  $J_j$  can be rearranged such that  $J_i$  is totally executed

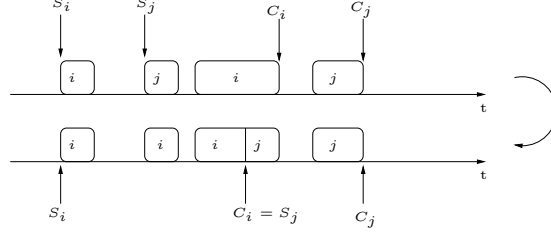


Figure 1. Proof of Property 1

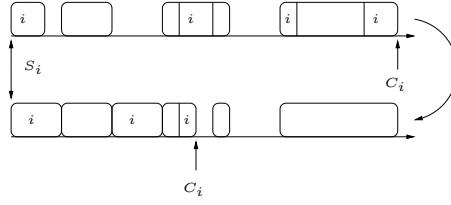


Figure 2. Proof of Property 2

before the start of  $J_j$ . Consequently the completion time of  $J_i$  is decreased and the starting time of  $J_j$  is increased. Thus the costs induced by the two jobs can only decrease (see Figure 1).

**Property 2.** *A sequence such that there is some idle time between the starting time and the completion time of a job is dominated.*

*Proof:* Consider a schedule where there is some idle time between the start of job  $J_i$  and its completion. The execution intervals of time of job  $J_i$  can be rearranged in order to fill the gaps between the start and the completion of  $J_i$ . In that manner,  $C_i$  may only decrease (see Figure 2).

For a pair of jobs  $(J_i, J_j)$ , we then have two possibilities: either  $S_i \leq S_j \leq C_j \leq C_i$  or  $S_i \leq C_i \leq S_j \leq C_j$ . In the first case, we say that  $J_j$  is *nested* in  $J_i$ . In the second case, we say that  $J_i$  and  $J_j$  are *separated*. The first property ensure a structure analogous to the nested parenthesis structure [Aho and Ullman, 1994]: for a given job  $J_j$ , each job that has started after  $S_j$ , has finished before  $C_j$ , and is therefore nested in  $J_j$ . According to the second property, there is no idle time between  $S_j$  and  $C_j$ . We call  $B_j$  the set of jobs that must be processed between  $S_j$  and  $C_j$  ( $J_j$  included). Since there is no idle time, we will call this set the block



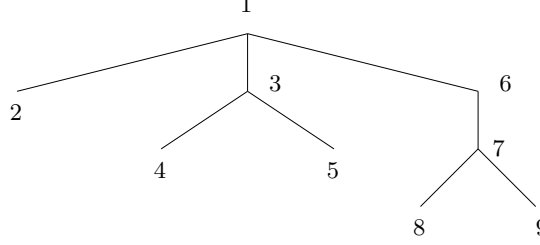
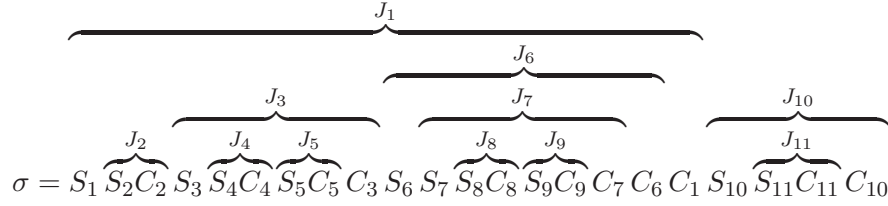


Figure 3. The date tree associated to the first *base-block* of  $\sigma$

associated with the job  $J_j$ . We finally denote by  $P_j$  the total processing time of the block  $B_j$ , which is equal to  $P_j = p_j + \sum_{J_k \in B_j} p_k$ .

A sequence, which has this nested parenthesis structure, is said *valid*. A *main block* is a block not nested by any other job. Thus a valid sequence can be divided into  $m$  main blocks which represent the jobs that are not *nested* by any other job. We call  $B_1 \dots B_m$  (in the order of the sequence) these main blocks. In the following section, we will show how to compute the cost for scheduling each of the main blocks.

Below is an example of a valid sequence of 11 jobs:



Sequence  $\sigma$  has two main blocks, one tied to  $J_1$  and one tied to  $J_{10}$ .

## 2. Solving the subproblem

### 2.1 Data Structure

According to the well-known connection between nested parenthesis expression and forest of trees, we use a tree to encode a main block: a leaf represents a job  $J_j$  such that the immediate successor in the sequence of  $S_j$  is  $C_j$  (therefore,  $C_j - S_j = p_j$ ). A node  $r$  with  $k$  sons represents a job  $J_r$  embracing the  $k$  sub-blocks represented by the  $k$  sons in their left-right order. Therefore the number of nodes in the tree corresponds to the number of jobs in the main block. The tree represented in figure 3 corresponds to the first main block of  $\sigma$ .

In each node  $r$ , we store:

- $P_r$  the total processing time between  $S_r$  and  $C_r$

- $f_r(t)$  the cost function for scheduling the block  $B_r$  such that  $C_r = t$  (and  $S_r = t - P_r$ ).

We are going to prove by induction that the information stored in each node can be derived from the information stored in the son nodes. Moreover, we also prove that the cost functions  $f_i$  are piecewise linear and convex. First, since a leaf represents a stand-alone job —say  $J_i$ — that is executed without preemption, its cost function is therefore equal to  $\max(\beta_i(t - d_i^c), \alpha_i(d_i^c - t))$ . Clearly, this function is piecewise linear and convex.

We then consider a subtree whose root encodes information about the job  $J_r$ . We rename  $\mathbb{T}_1 \dots \mathbb{T}_k$  the descendant trees of this root in the left-right order. The tree  $\mathbb{T}_i$  holds the cost function of  $B_i$ , which is piecewise linear and convex by the induction hypothesis, and the length  $P_i$  of block  $B_i$ . We first have  $P_r = p_r + \sum_{j=1}^k P_j$ . If we fix  $C_r$ ,  $S_r$  is also fixed ( $S_r = C_r - P_r$ ). The blocks  $B_1, \dots, B_k$ , in this order, have to be optimally scheduled within the time interval  $[S_r, C_r]$ , the sum of the idle periods in this interval being  $p_r$ . However, we need to know the cost of the block  $B_r$  when  $C_r$  varies. In the following, we propose a method to compute  $f_r(C_r)$  efficiently. The idea is first to get the ideal completion time of every sub-block when they are not constrained by  $S_r$  and  $C_r$  (In that case, there is no preemption, so we can solve the problem using the extension of [Garey et al., 1988] algorithm proposed by [Chrétienne and Sourd, 2003]): it provides a new block decomposition (see figure 4 - case a). Then, we show that for any  $C_r$ , the scheduling of  $B_1, \dots, B_k$  in  $[S_r, C_r]$  can be derived from the non-constrained schedule so that we finally have an algorithm to compute  $f_r$  from  $f_1, \dots, f_k$ .

## 2.2 Computation of a node

**2.2.1 Determining the sub-blocks of  $B_r$ .** We apply the algorithm of Chrétienne and Sourd to the sequence made of the blocks  $B_1 \rightarrow B_k$ : it provides the schedule  $B_1, \dots, B_k$  with the minimal cost. A new block structure is extracted: let  $B'_j$  be the concatenation of consecutive blocks  $B_u, \dots, B_v$  in this schedule such that there is no idle time between  $B_u$  and  $B_v$  but idle time before  $B_u$  and after  $B_v$ . The cost function of  $B'_j$  is  $f'_j(t) = \sum_{i=u}^v f_i(t - \sum_{w>i} p_w)$  (see figure 4 - case a).  $f'_j$  is piecewise linear and convex as a sum of piecewise linear and convex functions.

**2.2.2 Computing the contribution of a block.** We next have to add the two constraints which state that the jobs have to be executed after  $S_r$  and before  $C_r$ : setting  $S_r$  force the far left blocks to

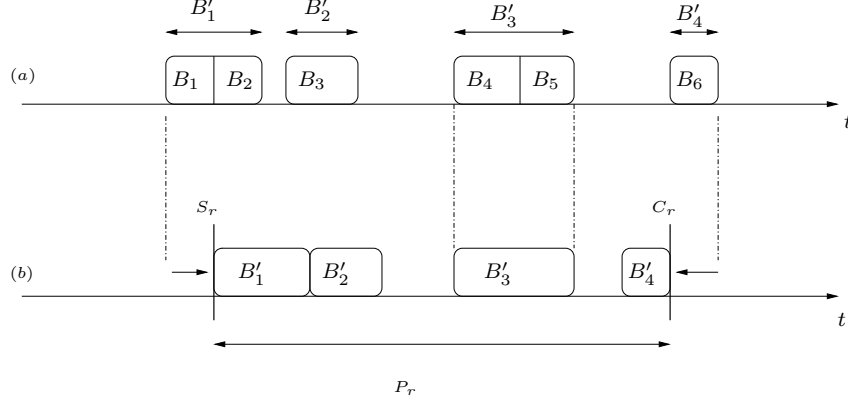


Figure 4. Optimal schedule without a completion time constraint, next constrained by  $C_r$ .

be right-shifted and setting  $C_r$  force the the far right blocks to be left-shifted (see figure 4 - case b). We then say a block is *critical* when it is constrained by the job  $J_r$ : a block  $B'_j$  is said to be *right-critical* (resp. *left-critical*), when there is no idle time between  $B'_j$  and  $C_r$  (resp. when there is no idle time between  $S_r$  and  $B'_j$ ). At any moment, a block  $B'_j$  is either left-critical, either *on time*, or right-critical; indeed, if a block is not critical, it is set at its due date in order to minimize its cost.

Formally, we have  $l$  blocks,  $B'_1 \dots B'_l$ , and we know the cost function  $f'_j(t)$  of each block. We denote by  $t_j$  the earliest time at which  $f'_j$  is minimum. We say that block  $B'_j$  become *on time* at time  $\tau_j = t_j + p_r + \sum_{i>j} P_i$  and  $B'_j$  become *right-critical*, at time  $\tau'_j = t_j + \sum_{i>j} P_i$ . While  $t > \tau_j$ ,  $B'_j$  is left critical, its cost is given by  $f'_j(t - p_r - \sum_{i>j} P_i)$ . Between  $\tau'_j < t \leq \tau_j$ ,  $B'_j$  is on time, its cost is given by  $f'_j(t_j)$ . Finally, when  $t \leq \tau'_j$ ,  $B'_j$  is right critical, its cost is given by  $f'_j(t - \sum_{i>j} P_i)$ .

In figure 4 - case b,  $B'_1$  and  $B'_2$  are left-critical,  $B'_3$  is on time and  $B'_4$  is right critical.

Eventually, we obtain the cost of the block  $B_r$  by adding all the block's contribution, which we have supposed piecewise linear and convex, and the earliness and tardiness cost of  $B_r$ . This sum is piecewise linear and convex too. We have thus proved the induction hypothesis stated in section 2.1.

**2.2.3 Complexity.** The cost functions  $f_1, \dots, f_k$  are piecewise linear and convex and we denote by  $\|f_1\|, \dots, \|f_k\|$ , their number of segments. Indeed, each cost function is fully described by the sorted list

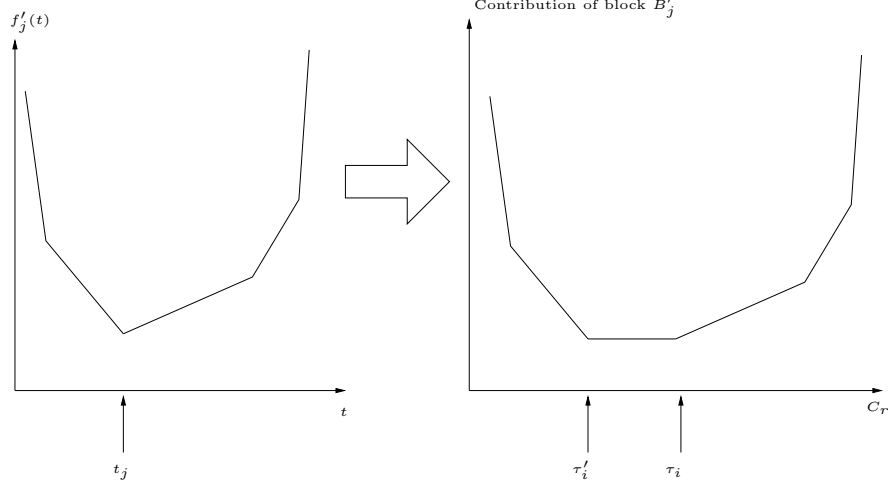


Figure 5. Cost function of a block  $B'_j$  and its contribution in the cost function  $B_r$ .

of its segments (each segment is represented by the coordinates of its endpoint). In [Chrétienne and Sourd, 2003], it is proved that the non constrained schedule can be computed in  $O(\sum_{i=1}^k \|f_i\| \cdot \log(\sum_{i=1}^k \|f_i\|))$ . The computing of the  $(B'_j)_{1 \leq j \leq l}$ , is then done in  $O(k)$  time and the sum of the number of segments of the  $B'_j$  is at most  $\sum_{i=1}^k \|f_k\|$ . To compute the contribution of a block  $B'_j$  in the calculus of  $B_r$ , one segment is added (see figure 5): finally, adding the contributions of each  $B'_j$  is done in  $O(\sum_{i=1}^k \|f_k\|)$ .

In conclusion,  $B_r$  (standing for the node  $r$ ) is computed in  $O(\sum_{i=1}^k \|f_i\| \cdot \log(\sum_{i=1}^k \|f_i\|))$  and has at most  $\sum_{i=1}^k \|f_k\| + k + 2$  (2 stands for the two segments added by the  $J_r$  earliness/tardiness function) segments.

### 2.3 Computing the minimum cost associated with a tree and solving the sub-problem

Consider a main block described by a tree with  $n'$  nodes, we prove by induction that the cost function stored at the root has less than  $3 \cdot n' - 1$  segments: each cost function stored at a leaf has two segments ( $3 \cdot 1 - 1$ ). Suppose that the root has  $k$  subtrees which represent  $n'_1, \dots, n'_k$  nodes such that  $\sum_{i=1}^k n'_i = n' - 1$  and each subtree has less than  $3 \cdot n'_i - 1$  segments. From the preceding section, we have clearly the maximal number of segments of  $B'_r$  :  $\sum_{i=1}^k 3 \cdot (n'_i - 1) + 2 + k$  which is less than  $3 \cdot n' - 1$ .

Every node of the tree has to be computed, starting at its leaves. Each node has at most  $3 \cdot n' - 1$  segments. In the preceding section, we have proved that its computation can be done in  $O(n' \cdot \log(n'))$ . Since there are  $n'$  nodes, the cost function of the main block can therefore be computed in  $O(n'^2 \cdot \log(n'))$ .

Suppose that we have  $m$  main blocks. We first have to compute their respective cost functions  $f_1, \dots, f_m$  (in left-to-right order): it can be done in  $O(n^2 \cdot \log(n))$ . To obtain the final schedule, we can again apply [Chrétienne and Sourd, 2003] algorithm. It's done in  $O(\sum_{i=1}^m \|f_i\| \cdot \log \sum_{i=1}^m \|f_i\|)$  which is equal to  $O(n \log(n))$ .

The global complexity of the algorithm is therefore  $O(n^2 \cdot \log(n))$ .

### 3. Conclusion and perspectives

We have introduced a variation of the one-machine earliness/tardiness scheduling problem and proposed a direct algorithm to solve the subproblem where the sequence of the starting and completion of the jobs is given. Future works could adress the general resolution of this problem. The search of lower bounds could be of interest in order to use the above subproblem in a branch and bound algorithm. Neighborhood search could be tried too. Actually, the nested parenthesis structure provide a convenient scheme in these two axes.

### References

- [Aho and Ullman, 1994] Aho, A. V. and Ullman, J. D. (1994). *Foundations of Computer Science*. W. H. Freeman.
- [Chrétienne and Sourd, 2003] Chrétienne, P. and Sourd, F. (2003). Pert scheduling with convex cost functions. *Theoretical Computer Science*, 292:145–164.
- [Garey et al., 1988] Garey, M., Tarjan, R., and Wilfong, G. (1988). One processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, 13:330 – 348.
- [Hendel and Sourd, 2003] Hendel, Y. and Sourd, F. (2003). Job-shop à deux tâches avec critères irréguliers. *Ecole d'Automne de Recherche Opérationnelle*.
- [Lenstra et al., 1977] Lenstra, J., Rinnooy Kan, A., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.
- [Sourd, 2002] Sourd, F. (2002). Scheduling a sequence of tasks with general completion costs. *European Journal of Operational Research*, to appear.

# THE SINGLE-MACHINE EARLINESS-TARDINESS SCHEDULING PROBLEM WITH DUE DATE ASSIGNMENT AND RESOURCE-DEPENDENT PROCESSING TIMES

George Steiner and Dvir Shabtay \*  
Management Science and Information Systems Area  
Michael G. DeGroote School of Business  
McMaster University, Hamilton, Ontario, Canada.

April 25, 2005

## Abstract

We study the earliness-tardiness scheduling problem on a single machine with due date assignment and controllable processing times. We analyze the problem with three different due date assignment methods and two different processing time functions. For each combination of these, we provide a polynomial-time algorithm to find the optimal job sequence, due date values and resource allocation minimizing an objective function which includes earliness, tardiness, due date assignment, makespan and total resource consumption costs.

*Keywords:* Single-machine scheduling; due date assignment; controllable processing times; resource allocation; earliness-tardiness scheduling.

## 1 Introduction

Meeting due dates has always been one of the most important objectives in scheduling and supply chain management. Customers demand that suppliers meet contracted delivery dates or face large penalties. For example, Slotnick and Sobel [24] cite contracts from the aerospace industry, which may impose tardiness penalties as high as one million dollars per day on subcontractors for aircraft components. The widespread use of Just-in-Time systems in industry made the early delivery of products also undesirable. This led to the introduction of earliness penalties, which may reflect additional storage or insurance costs, or costs of product deterioration over time. In the context of supply chains, we consider the supplier's system as having one stage, which means that it can be modeled by a single machine. While traditional scheduling models considered due dates as given by exogenous decisions (see [5] for a survey), in an integrated system they are determined by taking into account the system's ability to meet the quoted delivery dates. In order to avoid earliness-tardiness penalties, including the possibility of losing customers, companies are under increasing pressure to quote attainable delivery dates. At the same time, promising delivery dates too far into the future may not be acceptable to the customer or may force a company to offer price discounts in order to retain the business. Thus there is an important tradeoff between assigning relatively short due dates to customer orders and avoiding tardiness penalties. This is why an increasingly large number of recent studies have viewed due date assignment as part of the scheduling process, and showed how the ability to control due dates can be a major factor in improving system performance. In many scheduling problems, the job processing times can also be controlled by changing the allocation of resources to the jobs, which may result in further efficiencies. In this paper, we study a single-machine scheduling problem with an integrated objective function which includes earliness-tardiness penalties, resource consumption costs and the costs of due date assignment and schedule duration.

---

\*e-mails: shabtay@univmail.cis.mcmaster.ca and steiner@mcmaster.ca

Sequencing problems with controllable processing times have been studied extensively by researchers since 1980 (e.g., Vickson [25], Van Wassenhove and Baker [26], Janiak [13], Daniels [8], Alidaee and Ahmadian [2], Hoogeveen and Woeginger [10], Ng *et al.* [15]). A survey of results on this subject, up to 1990, is given by Nowicki and Zdrzalka [16]. In all of the above-mentioned studies of scheduling with controllable processing times, researchers assumed that the job processing time is a bounded linear function of the amount of resource allocated to the processing of the job, i.e., the *resource consumption function* is of the form

$$p_j(u_j) = \bar{p}_j - a_j u_j, \quad j = 1, \dots, n, \quad 0 \leq u_j \leq \bar{u}_j < \bar{p}_j/a_j, \quad (1)$$

where  $n$  is the number of non-preemptive jobs,  $p_j$  is the processing time of job  $j$ ,  $u_j$  is the amount of resource allocated to job  $j$ ,  $\bar{p}_j$  is the non-compressed processing time for job  $j$ ,  $\bar{u}_j$  is the upper bound on the amount of resource that can be allocated to job  $j$  and  $a_j$  is the positive compression rate of job  $j$ . For many resource allocation problems in physical or economic systems, however, they do not use a linear resource consumption function, since it fails to reflect the *law of diminishing marginal returns*. This law states that productivity increases at a decreasing rate with the amount of resource employed. In order to model this, other studies on scheduling with resource allocation assumed that the job processing time is a convex decreasing function of the amount of resource allocated to the processing of the job (e.g., Monma *et al.* [14], Shabtay [22], Shabtay and Kaspi [23]). For a convex resource consumption function, we assume the following relationship between the job processing time and the resource allocated to the job:

$$p_j(u_j) = \left( \frac{w_j}{u_j} \right)^k, \quad (2)$$

where  $w_j$  is a positive parameter, which represents the *workload* of the processing operation for job  $j$  and  $k$  is a positive constant. This resource consumption function has been used extensively in continuous resource allocation theory (e.g., Monma *et al.* [14], Scott and Jefferson [20], Armstrong *et al.* [3], Armstrong *et al.* [4], Shabtay [22] and Shabtay and Kaspi [23]). In fact, Monma *et al.* [14] pointed out that  $k = 1$  corresponds to many actual government and industrial operations and the  $k = 0.5$  case arises from VLSI (very large scale integration) circuit design, where the product of the silicon area (resource) and the square of time spent equals a constant value (the workload) for an individual job.

The objective in the problem we study is to determine the job sequence  $\pi^* \in \Pi$ , the set of due dates  $\mathbf{d}^* = (d_1^*, d_2^*, \dots, d_n^*)$  and the resource allocation  $\mathbf{u}^* = (u_1^*, u_2^*, \dots, u_n^*)$  which minimize a cost function that includes the costs of earliness, tardiness, due date assignment, makespan and resource consumption as given by the following equation:

$$Z(\pi, \mathbf{d}, \mathbf{u}) = \alpha \sum_{j=1}^n E_j + \beta \sum_{j=1}^n T_j + \gamma \sum_{j=1}^n d_j + \delta C_{\max} + \sum_{j=1}^n v_j u_j, \quad (3)$$

where  $\Pi$  is the set of all  $n!$  permutations of the  $n$  jobs,  $C_j$  is the completion time of job  $j$  in  $\pi$ ,  $E_j = \max(0, d_j - C_j)$  is the earliness of job  $j$ ,  $T_j = \max(0, C_j - d_j)$  is the tardiness of job  $j$ ,  $C_{\max} = \max_{j=1, \dots, n} C_j$  is the maximal completion time (makespan),  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $\delta$  are nonnegative parameters representing the cost of one unit of earliness, tardiness, due date and operation time, respectively, and  $v_j$  is the cost of one unit of resource allocated to job  $j$ .

A large variety of due date assignment methods have been studied in the literature. Recent surveys of these were given by Gordon *et al.* [11, 12]. In this paper, we study our problem with the three most frequent due date assignment methods:

- The *common* due date assignment method (usually referred to as CON) where all jobs are assigned the same due date, that is  $d_j = d$  for  $j = 1, \dots, n$ .
- The *slack* due date assignment method (usually referred to as SLK) where jobs are given an equal flow allowance that reflects equal waiting time (i.e., equal slacks), that is,  $d_j = p_j + slk$  for  $j = 1, \dots, n$ , where  $slk \geq 0$  is a decision variable.
- The *unrestricted* due date assignment method where each job can be assigned a different due date with no restrictions. (We will refer to this method as DIF in short.)

| Objective                              | Res. cons. fun.                     | Due Date Ass. St. | Complexity      | Reference |
|----------------------------------------|-------------------------------------|-------------------|-----------------|-----------|
| eq.(3) with $\gamma = \delta = 0$      | eq.(1) with $a_j = 1$               | CON               | $O(n^3)$        | [18], [2] |
| eq.(3) with $\delta = 0$               | eq.(1) with $a_j = 1$               | CON               | $O(n^3)$        | [7]       |
| eq.(3) with $\delta = 0$               | eq.(1) with $a_j = 1$               | SLK               | $O(n^3)$        | [7]       |
| eq.(3) with $\delta = 0$ and $u_j = u$ | $p_j(u) = \bar{p}_j \times (1 - u)$ | CON               | $O(n \log n)$   | [6]       |
| eq.(3) with $\delta = 0$ and $u_j = u$ | eq.(1) with $u_j = u$               | CON               | $O(n^2 \log n)$ | [15]      |
| eq.(3)                                 | eq.(1)                              | CON/SLK/DIF       | $O(n^3)$        | Theorem 1 |
| eq.(3)                                 | eq.(2)                              | CON/SLK/DIF       | $O(n \log n)$   | Theorem 2 |

Table 1: Summary of results

We study optimal schedules under these three due date assignment methods with both types of resource consumption function (eqs. (1)-(2)). For a given feasible resource allocation, which fixes the processing times and the makespan and resource consumption costs, our problem is reduced to finding a job sequence  $\pi$  and a set of due dates  $\mathbf{d} = (d_1, d_2, \dots, d_n)$  which minimize  $Z(\pi, \mathbf{d}) = \alpha \sum_{j=1}^n E_j + \beta \sum_{j=1}^n T_j + \gamma \sum_{j=1}^n d_j$ . Seidmann *et al.* [21] presented an  $O(n \log n)$  optimization algorithm to solve this problem for the DIF due date assignment method. Panwalkar *et al.* [17] and Adamopoulos and Pappis [1] showed that this problem can also be solved in  $O(n \log n)$  time for the CON and the SLK due date assignment methods, respectively.

There are several previous papers which combined due date assignment and continuous resource allocation decisions to minimize an objective which includes earliness and tardiness penalties (Panwalkar and Rajagopalan [18], Alidaee and Ahmadian [2], Cheng *et al.* [7], Biskup and Jahnke [6] and Ng *et al.* [15]). All of these papers assumed a linear resource consumption function as given by eq. (1) and most of them dealt only with the CON method. These results are summarized in the first five rows of Table 1, and we briefly review them next. For the case when  $a_j = 1$ ,  $j = 1, \dots, n$ , Panwalkar and Rajagopalan [18] presented an algorithm to find the job sequence, resource allocation and due date which minimize a cost function containing earliness, tardiness and resource consumption costs for the CON method. They proved that each job will be processed either with its non-compressed (maximal) or its most compressed (minimal) processing time, and reduced the problem to an assignment problem solvable in polynomial time ( $O(n^3)$ ). Alidaee and Ahmadian [2] extended Panwalkar and Rajagopalan's results to the case of identical parallel machines and solved the problem by reducing it to a transportation problem. Cheng *et al.* [7] also extended Panwalkar and Rajagopalan's research by adding the due date cost to the objective and by also solving the problem for the case of slack due date assignments (SLK). For the CON method, Biskup and Jahnke [6] studied the special case where the job processing times are jointly reducible by the *same proportional* amount, i.e., the case where  $a_j = \bar{p}_j$  and  $u_j = u$  for  $j = 1, \dots, n$ . They presented  $O(n \log n)$  optimization algorithms to minimize a cost function containing earliness, tardiness, resource consumption and due date assignment costs. Ng *et al.* [15] extended Biskup and Jahnke's results to the case where the job processing times are jointly reducible by the *same amount* of the resource, i.e., for the case where  $u_j = u$  for  $j = 1, \dots, n$ , and presented  $O(n^2 \log n)$  optimization algorithms for the same objective.

In most classical single-machine due date assignment problems where job processing times are fixed, one of the main properties is that an optimal schedule does not have any idle time. As a result, the makespan is identical for all job permutations. An important issue when dealing with controllable processing times, however, is that the makespan is no longer a constant value, it is a function of the resource allocation strategy. Therefore, it is quite surprising that previous studies that combined resource allocation with due date assignment did not include the makespan cost in the objective function.

Our paper extends the existing research on due-date assignment with controllable job processing times in several aspects: Since minimizing the makespan is one of the most important scheduling objectives, we include the makespan cost in our cost function; For the linear resource consumption function, we study the problems with the general function given in eq. (1), while previous studies dealt only with special cases of this function; We include a complete analysis of the three due date assignment methods with the convex resource consumption function given in eq. (2); Our algorithms for both resource consumption functions represent a *unified* solution approach for the different due date assignment methods. In the last two rows of Table 1, we include a summary of our main results.

The rest of the paper is organized as follows. In Section 2, we present some relevant preliminary results



for the reduced problems with fixed processing times. Section 3 presents an  $O(n^3)$  optimization algorithm to solve the problem for each of the three due date assignment methods assuming a linear resource consumption function. For the convex resource consumption function, we show in Section 4 that the optimal schedule can be determined in  $O(n \log n)$  time for all three due date assignment methods. Section 5 includes a numerical example, while the last section contains our concluding remarks.

## 2 Summary of Preliminary Results for Related Problems with Fixed Processing Times

Any given feasible resource allocation  $\mathbf{u}$  fixes the job processing times, the makespan and the resource consumption cost. Thus in this case, our problem is reduced to finding a job sequence  $\pi$  and a set of due dates  $\mathbf{d} = (d_1, d_2, \dots, d_n)$  which minimize  $Z(\pi, \mathbf{d}) = \alpha \sum_{j=1}^n E_j + \beta \sum_{j=1}^n T_j + \gamma \sum_{j=1}^n d_j$  for the three different due date assignment methods. In the following we present some earlier results obtained by Panwalkar *et al.* [17], Seidmann *et al.* [21] and Adamopoulos and Pappis [1] for the CON, DIF and SLK due date assignment methods, respectively. The following lemma is applicable for all three due date assignment methods.

**Lemma 1** *The optimal schedule does not include idle times with any of the three due date assignment strategies.*

As a result, we can conclude from Lemma 1 that

$$C_{[j]} = \sum_{i=1}^j p_{[i]}, \quad (4)$$

where  $[j]$  represents the job that is in the  $j^{th}$  position in schedule  $\pi$  for  $j = 1, 2, \dots, n$ .

### 2.1 Preliminary results for the CON due date assignment method

The following results are given by Panwalkar *et al.* [17] for the CON due date assignment method.

**Lemma 2** *For the CON due date assignment method, there exists an optimal due-date equal to  $C_{[l^*]}$ , where*

$$l^* = \min \left( \max \left( \left\lceil \frac{n \times (\beta - \gamma)}{\alpha + \beta} \right\rceil, 0 \right), n \right), \quad (5)$$

and  $C_0 = 0$  by definition.

Note that the value of  $l^*$ , given by eq. (5), is independent of the job processing times and the job sequence. Therefore, it is optimal for *any* job sequence and processing times. Thus for any  $\mathbf{u}$  and  $\pi$ , the following holds:

$$d_{[j]}^* = d^* = C_{[l^*]} = \sum_{j=1}^{l^*} p_{[j]} \quad \text{for } j = 1, \dots, n; \quad (6)$$

$$E_{[j]} = \sum_{i=j+1}^{l^*} p_{[i]}, \quad \text{for } j = 1, \dots, l^* - 1 \text{ and } E_{l^*} = 0; \quad (7)$$

$$T_{[j]} = \sum_{i=l^*+1}^j p_{[i]}, \quad \text{for } j = l^* + 1, \dots, n. \quad (8)$$

By substituting eqs. (6)-(8) into eq. (3), we get a new expression for our objective function under an optimal due date assignment strategy, denoted by  $\mathbf{d}^*(\pi, \mathbf{u})$ , for the CON due date assignment method:

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^{l^*} (\alpha(j-1) + \gamma n + \delta) \times p_{[j]} + \sum_{j=l^*+1}^n (\beta(n-j+1) + \delta) \times p_{[j]} + \sum_{j=1}^n v_{[j]} u_{[j]}. \quad (9)$$

## 2.2 Preliminary results for the SLK due date assignment method

Adamopoulos and Pappis [1] showed that the CON and the SLK methods have similar properties and presented the following result for the SLK due date assignment method.

**Lemma 3** *For the SLK due date assignment method, there exists an optimal slack allowance,  $slk^*$ , equal to  $C_{[l^*-1]}$ , where  $l^*$  is given by eq. (5).*

As a result, the following holds for any  $\mathbf{u}$  and  $\pi$ :

$$slk^* = C_{[l^*-1]} = \sum_{i=1}^{l^*-1} p_{[i]}; \quad (10)$$

$$d_{[j]}^* = p_{[j]} + slk^* = p_{[j]} + \sum_{i=1}^{l^*-1} p_{[i]} \quad \text{for } j = 1, \dots, n; \quad (11)$$

$$E_{[j]} = \sum_{i=j}^{l^*-1} p_{[i]}, \quad \text{for } j = 1, \dots, l^* - 1 \text{ and } E_{[l^*]} = 0; \quad (12)$$

$$T_{[j]} = \sum_{i=l^*}^{j-1} p_{[i]}, \quad \text{for } j = l^* + 1, \dots, n. \quad (13)$$

By substituting eqs. (10)-(13) into eq. (3) we get a new expression for our objective function under an optimal due date assignment strategy for the SLK due date assignment method:

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^{l^*-1} (\alpha j + \gamma(n+1) + \delta) \times p_{[j]} + \sum_{j=l^*}^n (\beta(n-j) + \gamma + \delta) \times p_{[j]} + \sum_{j=1}^n v_{[j]} u_{[j]}. \quad (14)$$

As we can see, there is a lot of similarity between the CON and SLK methods.

## 2.3 Preliminary results for the DIF due date assignment method

The DIF due date assignment method to minimize earliness, tardiness and due date assignment costs was studied by Seidmann *et al.* [21] and they presented the following lemma which defines the optimal due date assignment strategy for a given  $\pi$ . This lemma is also applicable for our problem with a given  $\pi$  and fixed  $\mathbf{u}$  and processing times.

**Lemma 4** *For a given  $\pi$  and fixed processing times, the optimal due date assignment strategy for the DIF due date assignment method is defined as follows: if  $\gamma \geq \beta$  then set  $d_j = 0$ , otherwise set  $d_j = C_j$  for  $j = 1, \dots, n$ .*

From Lemma 4, we can conclude that  $E_j = 0$  for  $j = 1, \dots, n$ . Therefore, with an optimal due date assignment strategy, our objective function becomes

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \begin{cases} \beta \sum_{j=1}^n C_j + \delta C_{\max} + \sum_{j=1}^n v_j u_j & \text{if } \gamma \geq \beta \\ \gamma \sum_{j=1}^n C_j + \delta C_{\max} + \sum_{j=1}^n v_j u_j & \text{if } \gamma < \beta. \end{cases}$$

By using (4) and  $C_{\max} = \sum_{j=1}^n p_{[j]}$ , this can be further written as

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^n (\epsilon(n-j+1) + \delta) \times p_{[j]} + \sum_{j=1}^n v_{[j]} u_{[j]}, \quad (15)$$

where  $\epsilon = \min(\beta, \gamma)$ .

### 3 Solution with a Linear Resource Consumption Function

In this section, we will show that with a linear resource consumption function the optimal schedule for all three due date assignment methods can be obtained in  $O(n^3)$  time. By solving eq. (1) for each  $u_{[j]}$  and substituting these into eqs. (9), (14) and (15), we obtain the following objective functions under an optimal due date assignment for the CON, SLK and DIF due date assignment methods, respectively.

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^{l^*} (\alpha(j-1) + \gamma n + \delta - v_{[j]}/a_{[j]}) \times p_{[j]} + \sum_{j=l^*+1}^n (\beta(n-j+1) + \delta - v_{[j]}/a_{[j]}) \times p_{[j]} + \sum_{j=1}^n \frac{v_{[j]}}{a_{[j]}} \bar{p}_{[j]}; \quad (16)$$

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^{l^*-1} (\alpha j + \gamma(n+1) + \delta - v_{[j]}/a_{[j]}) \times p_{[j]} + \sum_{j=l^*}^n (\beta(n-j) + \gamma + \delta - v_{[j]}/a_{[j]}) \times p_{[j]} + \sum_{j=1}^n \frac{v_{[j]}}{a_{[j]}} \bar{p}_{[j]}; \quad (17)$$

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^n (\epsilon(n-j+1) + \delta - v_{[j]}/a_{[j]}) \times p_{[j]} + \sum_{j=1}^n \frac{v_{[j]}}{a_{[j]}} \bar{p}_{[j]}. \quad (18)$$

Note that the value  $\sum_{j=1}^n \frac{v_{[j]}}{a_{[j]}} \bar{p}_{[j]}$  is a constant for any sequence of the jobs.

Let us define  $\varpi_{[j]}$  for the three due date assignment methods as follows.

$$\varpi_{[j]} = \begin{cases} \alpha \times (j-1) + \gamma n + \delta - v_{[j]}/a_{[j]} & \text{for } j = 1, \dots, l^* \\ \beta \times (n-j+1) + \delta - v_{[j]}/a_{[j]} & \text{for } j = l^* + 1, \dots, n \end{cases} \text{ for the CON method;} \quad (19)$$

$$\varpi_{[j]} = \begin{cases} \alpha j + \gamma(n+1) + \delta - v_{[j]}/a_{[j]} & \text{for } j = 1, \dots, l^* - 1 \\ \beta \times (n-j) + \gamma + \delta - v_{[j]}/a_{[j]} & \text{for } j = l^*, \dots, n \end{cases} \text{ for the SLK method;} \quad (20)$$

and

$$\varpi_{[j]} = \epsilon(n-j+1) + \delta - v_{[j]}/a_{[j]} \quad \text{for } j = 1, \dots, n \text{ for the DIF method.} \quad (21)$$

i.e.,  $\varpi_{[j]}$ ,  $j = 1, \dots, n$ , represents the cost of job  $[j]$  in the objective function. In other words,  $\varpi_{[j]}$  could be viewed as the *positional* penalty for the job in the  $j$ th position. In the following lemma, we present the optimal resource allocation as a function of the job sequence for all three due date assignment methods.

**Lemma 5** *Expressed as a function of the job sequence, the optimal resource allocation,  $\mathbf{u}^*(\pi)$ , for all three due date assignment methods can be determined as follows:*

$$u_{[j]}^* = \begin{cases} 0 & \text{if } \varpi_{[j]} < 0 \\ u_{[j]} \in [0, \bar{u}_{[j]}] & \text{if } \varpi_{[j]} = 0 \\ \bar{u}_{[j]} & \text{if } \varpi_{[j]} > 0 \end{cases} \text{ for } j = 1, \dots, n. \quad (22)$$

**Proof.** Substituting (1) for  $p_{[j]}$  into eqs. (16)-(18) for the three due date assignment methods and taking the derivative by  $u_{[j]}$ , we see that  $\frac{dZ(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u}))}{du_{[j]}} = -a_{[j]} \times \varpi_{[j]}$  for  $j = 1, \dots, n$ . Since  $a_{[j]}$  is a positive parameter, we can conclude that the sign of  $\frac{dZ(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u}))}{du_{[j]}}$  must be the opposite of the sign of  $\varpi_{[j]}$  in order to minimize  $Z$ . Therefore, if  $\varpi_{[j]} < 0$ , we should not allocate any resource to job  $[j]$ ; if  $\varpi_{[j]} > 0$ , we will allocate the maximal feasible amount of resource to job  $[j]$ ; and if  $\varpi_{[j]} = 0$ , any feasible resource allocation can be optimal. ■

**Lemma 6** *The optimal sequence, for all three due date assignment methods, can be determined by solving a linear assignment problem requiring  $O(n^3)$  time.*

**Proof.** For  $1 \leq i, j \leq n$ , let us define

$$\varpi_{ij} = \begin{cases} \alpha \times (i-1) + \gamma n + \delta - v_j/a_j & \text{for } i = 1, \dots, l^* \\ \beta \times (n-i+1) + \delta - v_j/a_j & \text{for } i = l^* + 1, \dots, n \end{cases} \text{ for the CON method;} \quad (23)$$

$$\varpi_{ij} = \begin{cases} \alpha i + \gamma(n+1) + \delta - v_j/a_j & \text{for } j = 1, \dots, l^* - 1 \\ \beta \times (n-i) + \gamma + \delta - v_j/a_j & \text{for } j = l^*, \dots, n \end{cases} \text{ for the SLK method;} \quad (24)$$

$$\varpi_{ij} = \epsilon(n-i+1) + \delta - v_j/a_j \text{ for the DIF method.} \quad (25)$$

It is clear that  $\varpi_{ij}$  represents the positional penalty if job  $j$  is assigned to position  $i$  in the schedule.

Using eq. (22), we obtain that the optimal length of job  $j$  if it is scheduled in position  $i$  is as follows:

$$p_{ij} = \begin{cases} \bar{p}_j & \text{if } \varpi_{ij} < 0 \\ p_j \in [\bar{p}_j - a_j \times \bar{u}_j, \bar{p}_j] & \text{if } \varpi_{ij} = 0 \\ \bar{p}_j - a_j \times \bar{u}_j & \text{if } \varpi_{ij} > 0 \end{cases}. \quad (26)$$

Therefore, if we define the value  $c_{ij}$  by

$$c_{ij} = \varpi_{ij} \times p_{ij} = \begin{cases} \varpi_{ij} \bar{p}_j & \text{if } \varpi_{ij} < 0 \\ 0 & \text{if } \varpi_{ij} = 0 \\ \varpi_{ij} (\bar{p}_j - a_j \times \bar{u}_j) & \text{if } \varpi_{ij} > 0 \end{cases}, \quad (27)$$

it represents the minimum possible cost resulting from assigning job  $j$  to position  $i$  in the sequence. Let us also define  $x_{ij} = 1$  if job  $j$  is assigned to position  $i$  and  $x_{ij} = 0$  otherwise. Our sequencing problem then can be formulated as the following linear assignment problem:

$$(P1) \quad \min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

s.t.

$$\sum_{i=1}^n x_{ij} = 1, \quad j = 1, \dots, n$$

$$\sum_{j=1}^n x_{ij} = 1, \quad i = 1, \dots, n$$

$$x_{ij} = 0 \text{ or } 1, \quad i, j = 1, \dots, n$$

The first set of constraints in the formulation assures that each job will be assigned only to one position, the second set assures that each position will be assigned only once, and the penalty for each assignment under an optimal resource allocation appears in the objective. It is well known that a linear assignment problem can be solved in  $O(n^3)$  time (see Papadimitriou and Steiglitz [19]). ■

The results of our analysis are summarized in the following optimization algorithm that solves our problem, for all three due date assignment methods, with a linear resource consumption.

**Algorithm 1** *The optimization algorithm for all three due date assignment methods, assuming a linear resource consumption function.*

*Step 1. (Apply this step only for the CON and the SLK methods) Calculate  $l^*$  by eq. (5).*

*Step 2. Calculate the  $c_{ij}$  values for each of the three due date assignment methods by eqs. (23)-(27).*

*Step 3. For each due date assignment method, solve the assignment problem (P1) to determine the optimal job sequence, and for each method denote the resulting optimal sequence by  $\pi^* = [1], [2], \dots, [n]$ .*

*Step 4. For each due date assignment method, allocate the resources according to eq. (22) and determine the optimal job processing times,  $p_j^*(u_j^*)$ , by eq. (1).*

*Step 5. For the CON method, assign the due date according to eq. (6), and for the SLK method, assign the due dates according to eqs. (10)-(11) with  $p_j = p_j^*(u_j^*)$ . For the DIF method, assign the due dates according to Lemma 4 with  $\pi = \pi^*$  and the job completion times calculated according to eq. (4) with  $p_j = p_j^*(u_j^*)$ .*

**Theorem 1** *Algorithm 1 solves the earliness-tardiness scheduling problem for each of the three due date assignment methods and linear resource consumption function in  $O(n^3)$  time.*

**Proof.** The correctness of the algorithm follows from Lemmas 1-6. Step 1 takes constant time; Steps 4 and 5 can be performed in linear time; Step 2 requires  $O(n^2)$  and Step 3  $O(n^3)$  time. Thus the overall computational complexity of the algorithm is  $O(n^3)$ , which is equal to the computational complexity of Step 3. ■

## 4 The Solution with a Convex Resource Consumption Function

By substituting eq. (2) into eqs. (9), (14) and (15), we obtain the following expressions for the objective functions under an optimal due date assignment strategy for the three due date assignment methods:

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^{l^*} (\alpha(j-1) + \gamma n + \delta) \times \left( \frac{w_{[j]}}{u_{[j]}} \right)^k + \sum_{j=l^*+1}^n (\beta(n-j+1) + \delta) \times \left( \frac{w_{[j]}}{u_{[j]}} \right)^k + \sum_{j=1}^n v_{[j]} u_{[j]} \quad (28)$$

for the CON method;

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^{l^*-1} (\alpha j + \gamma(n+1) + \delta) \times \left( \frac{w_{[j]}}{u_{[j]}} \right)^k + \sum_{j=l^*}^n (\beta(n-j) + \gamma + \delta) \times \left( \frac{w_{[j]}}{u_{[j]}} \right)^k + \sum_{j=1}^n v_{[j]} u_{[j]} \quad (29)$$

for the SLK method; and

$$Z(\pi, \mathbf{u}, \mathbf{d}^*(\pi, \mathbf{u})) = \sum_{j=1}^n (\epsilon(n-j+1) + \delta) \times \left( \frac{w_{[j]}}{u_{[j]}} \right)^k + \sum_{j=1}^n v_{[j]} u_{[j]} \quad (30)$$

for the DIF method.

In the following lemma, we determine the optimal resource allocation, denoted by  $\mathbf{u}^*(\pi)$ , as a function of the job sequence for each of the three due date assignment methods.

**Lemma 7** *The optimal resource allocation as a function of the job sequence,  $\mathbf{u}^*(\pi)$ , is:*

$$u_{[j]}^* = \left( \frac{k \times (\alpha(j-1) + \gamma n + \delta)}{v_{[j]}} \right)^{\frac{1}{k+1}} \times w_{[j]}^{\frac{k}{k+1}}, \quad \text{for } j = 1, \dots, l^*; \quad (31)$$

$$u_{[j]}^* = \left( \frac{k \times (\beta(n-j+1) + \delta)}{v_{[j]}} \right)^{\frac{1}{k+1}} \times w_{[j]}^{\frac{k}{k+1}}, \quad \text{for } j = l^* + 1, \dots, n, \quad (32)$$

for the CON method;

$$u_{[j]}^* = \left( \frac{k \times (\alpha j + \gamma(n+1) + \delta)}{v_{[j]}} \right)^{\frac{1}{k+1}} \times w_{[j]}^{\frac{k}{k+1}}, \quad \text{for } j = 1, \dots, l^* - 1; \quad (33)$$

$$u_{[j]}^* = \left( \frac{k \times (\beta(n-j) + \gamma + \delta)}{v_{[j]}} \right)^{\frac{1}{k+1}} \times w_{[j]}^{\frac{k}{k+1}}, \quad \text{for } j = l^*, \dots, n, \quad (34)$$

for the SLK method; and

$$u_{[j]}^* = \left( \frac{k \times (\epsilon(n-j+1) + \delta)}{v_{[j]}} \right)^{\frac{1}{k+1}} \times w_{[j]}^{\frac{k}{k+1}}, \quad \text{for } j = 1, \dots, n. \quad (35)$$

for the DIF method.

**Proof.** By taking the derivative of each of the objectives given by eqs. (28)-(30) with respect to  $u_{[j]}$ ,  $j = 1, \dots, n$ , equating it to zero and solving it for  $u_{[j]}^*$ , we obtain eqs. (31)-(35). Since each of the objectives is a convex function, eqs. (31)-(35) provide necessary and sufficient conditions for optimality. ■

By substituting eqs. (31)-(32) into eq. (28), eqs. (33)-(34) into eq. (29), and eq. (35) into eq. (30), we obtain a new *unified* expression for the cost function for the three due date assignment methods under an optimal resource allocation and due date assignment and as a function of the job sequence:

$$Z(\pi, \mathbf{u}^*(\pi), \mathbf{d}^*(\pi, \mathbf{u})) = \left( k^{\frac{-k}{k+1}} + k^{\frac{1}{k+1}} \right) \times \sum_{j=1}^n \theta_{[j]} \times \eta_j, \quad (36)$$

where

$$\theta_j = (w_j \times v_j)^{\frac{k}{k+1}}, \quad j = 1, \dots, n, \quad (37)$$

and

$$\eta_j = \begin{cases} (\alpha(j-1) + \gamma n + \delta)^{\frac{1}{k+1}}, & \text{for } j = 1, \dots, l^* \\ (\beta(n-j+1) + \delta)^{\frac{1}{k+1}}, & \text{for } j = l^* + 1, \dots, n \end{cases} \quad (38)$$

for the CON method;

$$\eta_j = \begin{cases} (\alpha j + \gamma(n+1) + \delta)^{\frac{1}{k+1}}, & \text{for } j = 1, \dots, l^* - 1 \\ (\beta(n-j) + \gamma + \delta)^{\frac{1}{k+1}}, & \text{for } j = l^*, \dots, n \end{cases} \quad (39)$$

for the SLK method; and

$$\eta_j = (\epsilon(n-j+1) + \delta)^{\frac{1}{k+1}}, \quad \text{for } j = 1, \dots, n. \quad (40)$$

for the DIF method.

In order to find the job sequence that minimizes  $Z(\pi, \mathbf{u}^*(\pi), \mathbf{d}^*(\pi, \mathbf{u}))$ , we have to optimally match the *positional* penalties  $\eta_j$  with the *job-dependent* costs  $\theta_j$ . The optimal matching is obtained by applying the following lemma.

**Lemma 8** *The optimal job sequence is obtained by matching the smallest  $\eta_j$  value to the job with the largest  $\theta_j$  value, the second smallest  $\eta_j$  value to the job with the second largest  $\theta_j$  value, and so on. The index of the  $\eta$  matched with  $\theta_j$  specifies the position of job  $j$  in the optimal sequence for  $j = 1, \dots, n$ .*

**Proof.** This follows from a well-known result in linear algebra about the minimization of a scalar product of two vectors (see Hardy *et al.* [9]). ■

The results of our analysis are summarized in the following optimization algorithm that solves our problem for all three due date assignment methods with the convex resource consumption given in eq. (2).

**Algorithm 2** *The optimization algorithm for all three due date assignment methods with a convex resource consumption function.*

- Step 1. (Apply this step only for the CON and the SLK methods) Calculate  $l^*$  by eq. (5).*
- Step 2. For each due date assignment method, calculate  $\theta_j$  and  $\eta_j$  for  $j = 1, \dots, n$  by eqs. (37)-(40).*
- Step 3. Sequence the jobs according to Lemma 8, and for each method denote the resulting optimal sequence by  $\pi^* = [1], [2], \dots, [n]$ .*
- Step 4. Allocate the resources according to eqs. (31)-(32) for the CON method, according to eqs. (33)-(34) for the SLK method and according to eq. (35) to the DIF method. For each method, determine the optimal job processing times,  $p_{[j]}^*(u_{[j]}^*)$  by eq. (2).*
- Step 5. For the CON method, assign the due date according to eq. (6) and for the SLK method assign the due dates according to eqs. (10)-(11), where  $p_{[j]} = p_{[j]}^*(u_{[j]}^*)$ . For the DIF method, assign the due dates according to Lemma 4 using  $\pi = \pi^*$  and calculate the job completion times by eq. (4) with  $p_{[j]} = p_{[j]}^*(u_{[j]}^*)$ .*

**Theorem 2** *Algorithm 2 solves the earliness-tardiness scheduling problem for each of the three due date assignment methods and a convex resource consumption function in  $O(n \log n)$  time.*

**Proof.** The correctness of the algorithm follows from Lemmas 1-4, and Lemmas 7-8. Step 1 takes constant time; Steps 2, 4 and 5 can be performed in linear time and Step 3 requires  $O(n \log n)$  time. Thus, the overall computational complexity of the algorithm is  $O(n \log n)$ , which is the computational complexity of Step 3. ■

## 5 Numerical Example

The following numerical example with  $n = 5$ ,  $\alpha = 1$ ,  $\beta = 2$ ,  $\gamma = 0.6$ ,  $\delta = 5$  and per unit resource cost as given in Table 2 is used to illustrate the optimization algorithms.

|       |    |    |    |   |    |
|-------|----|----|----|---|----|
| $j$   | 1  | 2  | 3  | 4 | 5  |
| $v_j$ | 20 | 14 | 27 | 8 | 24 |

Table 2: The per unit resource consumption penalties

Since the algorithm is similar for the three due date assignment methods, in the numerical example we will apply the algorithms only for the CON due date assignment method.

### 5.1 The linear resource consumption function

In this subsection, we assume that the processing time of each job follows the linear resource consumption function given in eq. (1). In Table 3, we specify the resource consumption function parameters for each job.

|             |    |    |    |   |    |
|-------------|----|----|----|---|----|
| $j$         | 1  | 2  | 3  | 4 | 5  |
| $a_j$       | 2  | 1  | 3  | 1 | 4  |
| $\bar{p}_j$ | 12 | 10 | 14 | 7 | 20 |
| $\bar{u}_j$ | 4  | 6  | 3  | 5 | 3  |

Table 3: The linear resource consumption parameters

We apply *Algorithm 1* to solve the problem for the CON due date assignment strategy.

*Step 1.*  $l^* = \min(\max(\left\lceil \frac{5 \times (2 - 0.6)}{1 + 2} \right\rceil, 0), 5) = 3$ .

*Step 2.* The  $\varpi_{ij}$  values were calculated according to eq. (23) and are given in Table 4. The  $p_{ij}$  values were calculated according to eq. (26) and are given in Table 5 (when  $\varpi_{ij} = 0$ , we arbitrarily set  $p_{ij} = \bar{p}_j - a_j \bar{u}_j$ ). The resulting  $c_{ij}$  values, which were calculated according to eq. (27), are given in Table 6.

*Step 3.* The costs for the solution of the assignment problem are highlighted in bold in Table 5 and the optimal job sequence is  $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3$ .

*Step 4.* The optimal resource allocations (obtained by eq. (22)) and the compressed processing times appear in Table 7.

*Step 5.*  $d^* = C_{[l^*]} = C_4 = 24$ .

|                  |    |    |    |    |   |
|------------------|----|----|----|----|---|
| $i \backslash j$ | 1  | 2  | 3  | 4  | 5 |
| 1                | -2 | -6 | -1 | 0  | 2 |
| 2                | -1 | -5 | 0  | 1  | 3 |
| 3                | 0  | -4 | 1  | 2  | 4 |
| 4                | -1 | -5 | 0  | 1  | 3 |
| 5                | -3 | -7 | -2 | -1 | 1 |

Table 4: The  $\varpi_{ij}$  values

|                  |    |    |    |   |   |
|------------------|----|----|----|---|---|
| $i \backslash j$ | 1  | 2  | 3  | 4 | 5 |
| 1                | 12 | 10 | 14 | 2 | 8 |
| 2                | 12 | 10 | 5  | 2 | 8 |
| 3                | 4  | 10 | 5  | 2 | 8 |
| 4                | 12 | 10 | 5  | 2 | 8 |
| 5                | 12 | 10 | 14 | 7 | 8 |

Table 5: The  $p_{ij}$  values

|                  |            |            |            |          |           |
|------------------|------------|------------|------------|----------|-----------|
| $i \backslash j$ | 1          | 2          | 3          | 4        | 5         |
| 1                | <b>-24</b> | -60        | -14        | 0        | 16        |
| 2                | -12        | <b>-50</b> | 0          | 2        | 24        |
| 3                | 0          | -40        | 5          | <b>4</b> | 32        |
| 4                | -12        | -50        | 0          | 2        | <b>24</b> |
| 5                | -36        | -70        | <b>-28</b> | -7       | 8         |

Table 6: Table  $c_{ij}$  values

|         |    |    |    |   |   |
|---------|----|----|----|---|---|
| $j$     | 1  | 2  | 3  | 4 | 5 |
| $[j]$   | 1  | 2  | 4  | 5 | 3 |
| $u_j^*$ | 0  | 0  | 0  | 5 | 3 |
| $p_j^*$ | 12 | 10 | 14 | 2 | 8 |

Table 7: The optimal resource allocations and job processing times

## 5.2 The convex resource consumption function

In this subsection, we assume that the processing time of each job follows the convex resource consumption function given in eq. (2). In Table 8, we specify the *workload* of each job and we also assume that  $k = 1$ .

|       |    |    |    |    |    |
|-------|----|----|----|----|----|
| $j$   | 1  | 2  | 3  | 4  | 5  |
| $w_j$ | 20 | 30 | 50 | 60 | 70 |

Table 8: The job workloads

We apply *Algorithm 2* to solve the problem for the CON due date assignment strategy.

*Step 1.*  $l^* = \min(\max(\lceil \frac{5 \times (2-0.6)}{1+2} \rceil, 0), 5) = 3$ .

*Step 2.* The  $\theta_j$  and  $\eta_j$  values obtained by eqs. (37)-(38) are given in Table 9.

*Step 3.* The optimal job sequence is  $3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 5$ .

*Step 4.* The optimal resource allocations (obtained by eqs. (31)-(32)) and processing times (calculated by eq. (2)) appear in Table 10.

*Step 5.*  $d^* = C_{[l^*]} = C_1 = 26.144$ .

|            |       |        |        |        |        |
|------------|-------|--------|--------|--------|--------|
| $j$        | 1     | 2      | 3      | 4      | 5      |
| $\theta_j$ | 20    | 20.494 | 36.742 | 21.909 | 40.988 |
| $\eta_j$   | 2.828 | 3      | 3.162  | 3      | 2.645  |

Table 9: The  $\theta_j$  and  $\eta_j$  values



|         |        |       |       |       |        |
|---------|--------|-------|-------|-------|--------|
| $j$     | 1      | 2     | 3     | 4     | 5      |
| $[j]$   | 3      | 2     | 1     | 4     | 5      |
| $u_j^*$ | 3.162  | 4.392 | 3.849 | 8.216 | 4.518  |
| $p_j^*$ | 13.000 | 6.826 | 6.328 | 7.302 | 15.496 |

Table 10: The optimal resource allocations and job processing times

## 6 Summary and Concluding Remarks

We have studied the single-machine earliness-tardiness scheduling problem where both the job processing times and the due dates are decision variables to be determined by the scheduler. We have presented unified polynomial-time algorithms under three of the most frequently used due date assignment methods for linear and convex resource consumption functions.

Future extensions of this research may include different objective functions, additional due date assignment strategies and different resource consumption functions.

## Acknowledgement

This research was partially supported by the Paul Ivanier Center for Robotics and Production Management, Ben-Gurion University of the Negev. Partial support by the Natural Sciences and Engineering Research Council of Canada under Grant No. 041798 is also gratefully acknowledged.

## References

- [1] Adamopoulos, G.I., and Pappis, C.P., 1996, Single Machine Scheduling With Flow Allowances, *Journal of the Operational Research Society*, **47**, 1280-1285.
- [2] Alidaee, B., and Ahmadian, A., 1993, Two Parallel Machine Sequencing Problems Involving Controllable Job Processing Times, *European Journal of Operational Research*, **70**, 335-341.
- [3] Armstrong, R., Gu, S., and Lei, L., 1995, An Algorithm for the Two-Resource Allocation Problem with a Non-differentiable Convex Objective Function, *Journal of the Operational Research Society*, **46**, 116-122.
- [4] Armstrong, R., Gu, S., and Lei, L., 1997, Solving a Class of Two-Resource Allocation Problem by Equivalent Load Method, *Journal of the Operational Research Society*, **48**, 818-825.
- [5] Baker, K.R. and Scudder, G.D. 1990, Sequencing With Earliness and Tardiness Penalties: A Review, *Operations Research*, **38**, 22-36.
- [6] Biskup, D., and Jahnke, H., 2001, Common Due Date Assignment for Scheduling On a Single Machine with Jointly Reducible Processing Times, *International Journal of Production Economics*, **69**, 317-322.
- [7] Cheng, T.C.E., Oğaz, C., and Qi, X.D., 1996, Due-Date Assignment and Single Machine Scheduling with Compressible Processing Times, *International Journal of Production Economics*, **43**, 29-35.
- [8] Daniels, R.L., 1990, A Multi-Objective Approach to Resource Allocation in Single Machine Scheduling, *European Journal of Operational Research*, **48**, 226-241.
- [9] Hardy, G.H., Littlewood, J.E., and Polya, G., 1934, *Inequalities*, Cambridge University Press, NY.
- [10] Hoogeveen, H., and Woeginger, G.J., 2002, Some Comments on Sequencing with Controllable Processing Times, *Computing*, **68**, 181-192.
- [11] Gordon, V., Proth, J.M., and Chu, C.B., 2002, A Survey of the State-of-the-Art of Common Due Date Assignment and Scheduling Research, *European Journal of Operational Research*, **139**, 1-25.
- [12] Gordon, V., Proth, J.M., and Chu, C.B., 2002, Due Date Assignment and Scheduling: SLK, TWK and Other Due Date Assignment Models, *Production Planning and Control*, **13** (2), 117-132.

- [13] Janiak, A., 1987, One-Machine Scheduling with Allocation of Continuously-Divisible Resource and with No Precedence Constraints, *Kybernetika*, **23**(4), 289-293.
- [14] Monma, C.I., Schrijver, A., Todd, M.J., and Wei, V.K., 1990, Convex Resource Allocation Problems on Directed Acyclic Graphs: Duality, Complexity, Special Cases and Extensions. *Mathematics of Operations Research*, **15**, 736-748.
- [15] Ng, C.T.D., Cheng, T.C.E., Kovalyov, M.Y., and Lam, S.S., 2003, Single Machine Scheduling with a Variable Common Due Date and Resource-Dependent Processing Times, *Computers and Operations Research*, **30**, 1173-1185.
- [16] Nowicki, E., and Zdrzalka, S., 1990, A Survey of Results for Sequencing Problems with Controllable Processing Times, *Discrete Applied Mathematics*, **26**, 271-287.
- [17] Panwalkar, S.S., Smith, M.L., and Seidmann, A., 1982, Common Due Date Assignment to Minimize Total Penalty for the One Machine Scheduling Problem, *Operations Research*, **30**, 391-399.
- [18] Panwalkar, S.S., and Rajagopalan, R., 1992, Single-Machine Sequencing with Controllable Processing Times, *European Journal of Operational Research*, **59**, 298-302.
- [19] Papadimitriou, C.H., and Steiglitz, K., 1982, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.
- [20] Scott, S.C., and Jefferson, T.R., 1995, Allocation of Resources in Project Management, *International Journal of Systems Science*, **26**(2), 413-420.
- [21] Seidmann, A., Panwalkar, S.S., Smith, M.L., and 1981, Optimal Assignment of Due Dates for a Single Processor Scheduling Problem, *International Journal of Production Research*, **19**, 393-399.
- [22] Shabtay, D., 2004, Single and a Two-Resource Allocation Algorithms for Minimizing the Maximal Lateness in a Single Machine-Scheduling Problem, *Computers and Operations Research*, **31**(8), 1303-1315.
- [23] Shabtay, D., and Kaspi, M., 2004, Minimizing the Total Weighted Flow Time in a Single Machine with Controllable Processing Times, *Computers and Operations Research*, **31**(13), 2279-2289.
- [24] Slotnick, S.A. and Sobel, M.J. 2005, Manufacturing lead-time rules: Customer retention versus tardiness costs, *European Journal of Operational Research*, **169**, 825-856.
- [25] Vickson, R.G., 1980, Two Single Machine Sequencing Problems Involving Controllable Job Processing Times, *AIIE Transactions*, **12** (3), 258-262.
- [26] Van Wassenhove, L., and Baker, K.R., 1982, A Bicriterion Approach to Time/Cost Trade-offs in Sequencing, *European Journal of Operational Research*, **11**, 48-54.

# A BRANCH-AND-BOUND ALGORITHM FOR SINGLE MACHINE TOTAL TARDINESS PROBLEM WITH RELEASE DATES

Ceyda Oğuz, Eda Yücel.

*Department of Industrial Engineering, Koç University, Rumeli Feneri Yolu, 34450 Sarıyer, İstanbul, Turkey*

This article presents the preliminary results on a new branch-and-bound algorithm proposed for the single machine total tardiness problem with arbitrary release dates. The algorithm relies on dominance properties from the literature as well as a new branching condition. A new lower bounding technique for the problem is also introduced. The performance of the algorithm is compared to the performance of one of the algorithms in the literature. These preliminary results show that the proposed approach is very promising as an efficient and effective exact algorithm for the problem.

Key words: Machine scheduling; Single machine; Total tardiness

## 1. PROBLEM DEFINITION AND LITERATURE REVIEW

In this paper, we consider the following scheduling problem: A set of  $n$  independent jobs has to be processed on a single machine. We assume that the single machine is available from time  $t=0$ , and it can process at most one job at a time. Each job  $j$  is characterized by its release date,  $r_j$ , its processing time,  $p_j$ , and its due date,  $d_j$ , ( $j = 1, \dots, n$ ). Hence, the processing of each job  $j$  can commence only after its release date,  $r_j$ , and it has to be completed at or before its due date,  $d_j$ , ( $j = 1, \dots, n$ ). Furthermore, preemptions are not allowed, that is, a job  $j$  has to be processed on the machine for an uninterrupted period of  $p_j$  units of time ( $j = 1, \dots, n$ ). Our objective is to find a schedule  $S$  which minimizes the total tardiness of all jobs,  $\sum T_j(S)$ , where  $T_j$  is the tardiness of job  $j$  and is defined as  $T_j = \max(C_j(S) - d_j, 0)$ , and  $C_j(S)$  is the completion time of job  $j$  under schedule  $S$  ( $j = 1, \dots, n$ ). Our problem can be denoted by  $1|r_j|\sum T_j$  by using the well-known three-field notation (see, for example, Błażewicz *et al.* (2001), Pinedo (2002)).

The  $1|r_j|\sum T_j$  problem has been shown to be NP-hard by Rinnooy Kan (1976). Erschler *et al.* (1983) have proved a dominance property for scheduling jobs on a single machine with unequal release dates. Later, Chu and Portmann (1989, 1991) have provided a dynamic priority rule for jobs and proposed heuristic solution methods.

Chu (1992) has proved additional dominance properties and proposed a lower bound for the problem. The branch-and-bound algorithm proposed by Chu solves hard problems up to 30 jobs and relatively easy problems (with equal release dates) up to 230 jobs.

Akturk (1995) has considered the same problem and proposed new dominance properties which are used during branching. He has also provided a tighter lower bound than that of Chu. However, the algorithm was not implemented so there are no comparisons of his algorithm with that of Chu (1992).

Most recently, Baptiste *et al.* (2004) have presented a new branch-and-bound procedure based on constraint propagation for the problem. They have introduced new lower bounds and generalized some well-known dominance properties.

In this paper, we propose a new branch-and-bound algorithm which is based on the branching and bounding techniques of Chu and Akturk. We describe this branch-and-bound algorithm in the next section. We also introduce a new branching condition for the algorithm in Section 2.1. We then present our preliminary results on the performance of the algorithm compared to the performance of the branch-and-bound algorithm of Chu in Section 3. We note that our computational experiments are continuing to compare the performance of our algorithm with that of Baptiste *et al.*'s algorithm.

## 2. THE BRANCH-AND-BOUND ALGORITHM

### 2.1 Branching

In this section, we present several branching strategies from the literature. The following theorem is due to Chu (1992) and was also used by Akturk (1995).

**Theorem 1(Chu (1992) - Akturk (1995)):** For any two jobs  $i$  and  $j$  with  $p_i \leq p_j$ , assuming both are ready at the current time  $t^c$ , the following rules minimize the total tardiness:

1. for  $p_i \neq p_j$ ,  
if  $d_i - p_j > t^c$ , use the Earliest Due Date (EDD) rule,  
otherwise, use the Shortest Processing Time (SPT) rule,
2. for  $p_i = p_j$ , use EDD.

The job that can be sequenced at the current time  $t^c$  according to Theorem 1 is called desirable job at time  $t^c$ . Theorem 1 states the sufficient condition for job  $i$  to precede job  $j$  at current time  $t^c$ , but it is not necessary condition for job  $i$  to precede job  $j$  in an optimal sequence. However, the following definition, which is called as the global dominance rule, states the necessary and the sufficient condition for job  $i$  to precede job  $j$  in an optimal sequence (Chu 1992).

**Definition 1:** If job  $i$  dominates job  $j$  for every  $t \geq t^c$ , then this unconditional ordering is called a global dominance for the  $1|r_j|\sum T_j$  problem and denoted by  $i \rightarrow j$ .

For our algorithm and the strategies presented below, the following sets are defined:

- $S(t^c)$  is the set of scheduled jobs until time  $t^c$ .
- $A(t^c)$  is the set of available and unscheduled jobs at time  $t^c$ , i.e.  $A(t^c) = \{i | r_i \leq t^c\} - S(t^c)$ .
- $B(t^c)$  is the set of unavailable and unscheduled jobs at time  $t^c$ , i.e.  $B(t^c) = \{k | r_k > t^c\}$ .
- Job  $j$  is the desirable job at time  $t^c$  due to Theorem 1.
- $D(t^c)$  is the set of unavailable and unscheduled jobs at time  $t^c$  that are more desirable than job  $j$  at their respective release dates,  $r_k$ .

Each node in the branch-and-bound tree represents a job and the path from the root node to a child node defines a partial schedule starting with the job represented by the root node and ending with the job represented by the last node in the path. In addition to the desirable job, which is defined by Theorem 1 above and is a candidate to be the next job in the sequence, there might be other unscheduled jobs to be sequenced as the next job in order to reach an improved sequence. The following corollaries define the conditions for scheduling such a job at the current time. Therefore, these corollaries constitute the

branching conditions for the nodes in the branch-and-bound tree. We note that while Corollaries 1, 2, and 4 are due to Akturk (1995), Corollary 3 describes a new branching strategy that we introduce.

**Corollary 1 (Akturk 1995) (Branching Condition 1):** For the desirable job  $j$  at time  $t^c$ , and for any job  $i \in A(t^c)$ , if  $p_i < p_j$  and  $t^c + p_i > d_i - p_j$ , then the current schedule may be improved by scheduling job  $i$  first.

**Corollary 2 (Akturk 1995) (Branching Condition 2):** For a desirable job  $j$  scheduled at time  $t^c$ , and for a job  $k \in B(t^c)$  that becomes available during the processing of job  $j$ , i.e.  $t^c + p_j > r_k$ , scheduling job  $k$  first may improve the schedule, if  $t^c + \min_{i \in A(t^c)}(p_i) > r_k$ , and either

1.  $p_k \geq p_j$ ,  $d_j - p_k \geq r_k$  and  $d_k \leq d_j$  (due to EDD rule), or
2.  $p_k < p_j$ , and  
either  $d_k - p_j \geq r_k$  and  $d_k \leq d_j$  (due to EDD rule),  
or  $d_k - p_j < r_k$  (due to SPT rule).

**Corollary 3 (Branching Condition 3):** For the desirable job  $j$  at time  $t^c$ , and for any job  $i \in A(t^c)$  that fails the Branching Condition 1, the current schedule may still be improved by scheduling job  $i$  first, if

1.  $p_i \neq p_j$ , and
2.  $t^c + \sum_{i \in A(t^c)} p_i > \min_{k \in B(t^c)}(r_k)$ , and
3. Either  $p_j > \min_{k \in B(t^c)}(p_k)$ ,  
or  $d_j > \min_{k \in B(t^c)}(d_k)$ , and
4. Either  $t^c + p_j < \max_{m \in D(t^c)}(r_m)$ ,  
or  $t^c + p_j \geq \max_{m \in D(t^c)}(r_m)$ , and  $p_i < p_j$ .

**Corollary 4 (Akturk 1995) (Branching Condition 4):** For the desirable job  $j$  at time  $t^c$ , which is also the desirable job compared to job  $k \in B(t^c)$  at time  $r_k$ , the current schedule may still be improved by scheduling job  $k$  first, if

1.  $t^c + \min_{i \in A(t^c)}(p_i) > r_k$ , and
2.  $r_k < \max_{m \in D(t^c)}(r_m)$ , and
3. Either  $p_j > p_k$  and  $t^c + p_j > r_k + p_k$ ,  
or  $p_j < p_k$  and  $t^c + p_j < \max_{m \in D(t^c)}(r_m)$ .

Therefore, at current time in order for a job to be scheduled next, it should either be a desirable job or a job proposed by one of the given branching conditions. In addition to this, if a job is supposed to be scheduled according to Branching Condition 1, 3 or 4 at current time, then it means that the desirable job of the current time cannot be scheduled just after that job. This is stated in the following lemma.

**Lemma 1 (Akturk 1995):** For the desirable job  $j$  at time  $t^c$ , any job  $i \in A(t^c)$  or job  $k \in B(t^c)$  that is scheduled due to either Branching Condition 1 or Branching Condition 3 or Branching Condition 4 cannot immediately precede job  $j$  in an optimal schedule.

## 2.2 Bounding

A lower bound for  $1|r_j|\sum T_j$  can be found by relaxing the problem to its preemptive version. However,  $1|r_j, \text{prmp}|\sum T_j$  is also NP-hard (Chu (1992)). Chu (1992) proposes to use SRPT (Shortest Remaining Processing Time First) rule with due dates ordered in non-decreasing order to generate a lower bound for the total tardiness problem. When this technique is combined with the following theorem, we get a tighter bound.

**Theorem 2 (Akturk (1995)):** For  $S'$ , which is the set of unscheduled jobs at time  $t^c$ , and for the desirable job  $j$  at time  $t^c$ , if either  $j \rightarrow i$  or  $t^c + p_i \leq d_i - p_j$  and  $d_j \leq d_i$  or  $r_i \geq t^c + p_j$  for every job  $i \in S'$ , then job  $j$  will be the first job in the remaining sequence when the preemption is allowed.

The upper bound in our algorithm is calculated by using the heuristic algorithm NDPRTT (Nondelay Schedule and Theorem 1) (Chu and Portmann (1991)). The algorithm runs as follows. At each iteration, the job which can be processed the earliest with the smallest PRTT value (if any tie occurs, the one with the smallest processing time is selected) is scheduled. PRTT value for a job  $i$  at time  $t^c$  is calculated as

$$\text{PRTT}(i, t^c) = b_i(t^c) + \max(b_i(t^c) + p_i, d_i),$$

where  $b_i(t^c)$  is the earliest starting time of job  $i$  at time  $t^c$  ( $b_i(t^c) = \max(t^c, r_i)$ ).

## 2.3 The Branch-and-Bound Algorithm

Our proposed branch-and-bound algorithm can be summarized as follows: Each node is defined by a partial sequence  $P$  of jobs scheduled from the beginning of the schedule. Initially,  $P$  is empty. Each descendant node is obtained by adding, behind the partial sequence  $P$ , a new job  $i$  chosen among the unscheduled jobs. The choice of job  $i$  is performed such that job  $i$  does not violate Lemma 1 and is either a desirable job at the completion time of the partial sequence  $P$ , or satisfies one of the Branching Conditions 1 to 4. The algorithm uses the breadth-first search and is based on only forward sequencing. Although Chu's algorithm based on both forward and backward sequencing, by using the new dominance properties, our algorithm eliminates more nodes in the branch-and-bound tree than that of Chu's.

## 3. COMPUTATIONAL EXPERIMENTS AND RESULTS

We implemented our branch-and-bound algorithm in C++. The computational experiments were performed on Pentium M 1.50 GHz processor with 248 MB of RAM. We generated the data according to the scheme given by Chu (1992). The experiment consists of 120 problems with 20 jobs and 120 problems with 30 jobs. Each problem is generated at random from three uniform distributions of  $r_i$ ,  $p_i$ , and  $d_i - (r_i + p_i)$ . The distribution of  $p_i$  is always between 1 and 10. The distribution of  $r_i$  is between 0 and  $\alpha \times \sum p_i$ . The distribution of  $d_i - (r_i + p_i)$  is between 0 and  $\beta \times \sum p_i$ . Four values for  $\alpha$ , and three values for  $\beta$  were combined to produce 12 problem sets, each containing 10 problems with 20 jobs and 10 problems with 30 jobs.

Tables 1-3 show the results of the problems with 20 jobs, respectively, for the number of nodes considered, the computation time (in seconds), and the mean deviation of lower bounds of nodes considered from the optimum ( $\sum_{i=1}^m (\text{OPT} - \text{LB}_i) / \text{OPT} / m = (\text{OPT} - \sum_{i=1}^m \text{LB}_i / m) / \text{OPT}$ , where  $\text{OPT}$  and  $\text{LB}_i$  are, respectively, the optimum value of the total tardiness for the problem and the lower bound of node  $i$ , and  $m$  is the number of nodes considered) for each problem set. In each table, the minimum, maximum and average values of these measures in the first, the second and the third columns for three different values of  $\beta$  are provided, respectively. In Table 3, we also report in parenthesis the minimum, the maximum and the average tardiness values for the set of 10 problems generated when we obtain the optimum solution. Tables 4-6 are analogous to Tables 1-3 for problems with 30 jobs.

When we analyze the results we observe that the number of nodes considered and consecutively the computation time for the algorithm increase for the difficult cases of the problem ( $\alpha=0.5$  and  $\beta=0.5$ ). This is in line with the observations made by Chu (1992). We note that this difficulty is increased with the increase in the number of jobs. From Tables 3 and 6, we can also notice that the lower bounds are quite successful for these difficult problems as the gap is not much larger compared to the other cases. These

preliminary results show that our algorithm is an alternative to Chu's algorithm (1992) and it efficiently solves problems involving 30 jobs. We remark from the results of Tables 1 and 4 that our algorithm has the potential to be more efficient than that of Chu's since we branch much less in the algorithm. This will be verified in our ongoing computational experiments for larger number of jobs.

## 4. CONCLUSIONS

In this paper we presented a branch-and-bound algorithm for  $1|r_j|\sum T_j$  problem. The algorithm is a combination of ideas from the literature, namely the branching conditions of Akturk (1995) and the bounding idea of Chu (1992), as well as a new branching condition we proposed which improves the performance of the algorithm. Our results indicate that the algorithm can be more efficient than that of Chu's but this should be verified with extended computational experiments. We will also consider the ideas presented in Baptiste *et al.* (2004) to improve our algorithm. We note that the branch-and-bound algorithm developed by Baptiste *et al.* is different from ours in its basic content. Nevertheless, their lower bounding algorithm may be integrated into ours in order to obtain better lower bounds, which might however increase the time complexity.

## REFERENCES

- M.S. Akturk (1995). An exact approach to minimize total tardiness on a single machine subject to release dates. Presented at Combinatorial Optimization Workshop.
- Ph. Baptiste, J. Carlier, and A. Jouglet (2004). A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 158:595-608.
- J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Weglarz (2001). *Scheduling Computer and Manufacturing Processes*, 2<sup>nd</sup> edition, Springer-Verlag, Berlin.
- C. Chu (1992). A branch-and-bound algorithm to minimize total tardiness with unequal release dates. *Naval Research Logistics*, 39:265–283.
- C. Chu and M. Portmann (1989). Minimisation de la somme des retards pour les probl`emes d'ordonnancement `a une machine. *Rapport de Recherche*.
- C. Chu and M. Portmann (1991). Some new efficient methods to solve  $n|1|r_i|\sum t_i$  scheduling problem. *European Journal of Operational Research*, 58:404-413.
- C. M. J. Erschler, G. Fontan and F. Roubellat (1983). A new dominance concept in scheduling  $n$  jobs on a single machine with ready times and due dates. *Operations Research*, 31:114–127.
- M. Pinedo (2002). *Scheduling Theory, Algorithms and Systems*, 2<sup>nd</sup> edition, Prentice Hall, New Jersey.
- A.H.G. Rinnooy Kan (1976). *Machine Sequencing Problem: Classification, Complexity and Computation*, Nijhoff, The Hague.

Table 1. The number of nodes considered (n=20)

| $\beta$  |      |     |      |      |     |       |      |     |       |
|----------|------|-----|------|------|-----|-------|------|-----|-------|
|          | 0.05 |     |      | 0.25 |     |       | 0.50 |     |       |
| $\alpha$ | min. | max | avg. | min. | max | avg.  | min. | max | avg.  |
| 0.0      | 21   | 35  | 25.2 | 21   | 143 | 65.9  | 21   | 177 | 78.5  |
| 0.5      | 21   | 147 | 65.0 | 21   | 258 | 157.8 | 37   | 689 | 398.5 |
| 1.0      | 22   | 125 | 47.6 | 21   | 138 | 69.4  | 21   | 45  | 30.2  |
| 1.5      | 21   | 30  | 25.3 | 21   | 60  | 30.3  | 21   | 40  | 26.2  |

Table 2. The computation time in seconds (n=20)

| $\beta$  |      |     |      |      |     |      |      |     |      |
|----------|------|-----|------|------|-----|------|------|-----|------|
|          | 0.05 |     |      | 0.25 |     |      | 0.50 |     |      |
| $\alpha$ | min. | max | avg. | min. | max | avg. | min. | max | avg. |
| 0.0      | 0    | 1   | 0.5  | 0    | 5   | 1.1  | 0    | 2   | 0.9  |
| 0.5      | 0    | 3   | 1.0  | 0    | 27  | 5.7  | 0    | 165 | 35.5 |
| 1.0      | 0    | 25  | 4.9  | 1    | 6   | 1.9  | 0    | 33  | 15.1 |
| 1.5      | 0    | 2   | 0.7  | 0    | 2   | 0.3  | 0    | 1   | 0.3  |

Table 3. Mean deviation of lower bounds from optimum (n=20)

| $\beta$  |                |                 |                  |                |                |                  |                |                |                  |
|----------|----------------|-----------------|------------------|----------------|----------------|------------------|----------------|----------------|------------------|
|          | 0.05           |                 |                  | 0.25           |                |                  | 0.50           |                |                  |
| $\alpha$ | min.           | max             | avg.             | min.           | max            | avg.             | min.           | max            | avg.             |
| 0.0      | 0.062<br>(422) | 0.070<br>(626)  | 0.065<br>(503.2) | 0.024<br>(168) | 0.091<br>(647) | 0.042<br>(353.4) | 0.080<br>(118) | 0.120<br>(373) | 0.92<br>(259.3)  |
| 0.5      | 0.010<br>(264) | 0.060<br>(483)  | 0.021<br>(302.7) | 0.000<br>(162) | 0.137<br>(331) | 0.070<br>(248.2) | 0.000<br>(11)  | 0.175<br>(176) | 0.068<br>(74.0)  |
| 1.0      | 0.000<br>(0)   | 0.100<br>(238)  | 0.070<br>(92.4)  | 0.000<br>(5)   | 0.137<br>(387) | 0.070<br>(208.1) | 0.000<br>(32)  | 0.175<br>(266) | 0.164<br>(128.7) |
| 1.5      | 0.060<br>(257) | 0.096<br>(1040) | 0.066<br>(470.5) | 0.058<br>(221) | 0.127<br>1050  | 0.090<br>(489.0) | 0.076<br>(20)  | 0.158<br>(477) | 0.099<br>(264.5) |

Table 4. The number of nodes considered (n=30)

| $\beta$  |      |      |        |      |       |        |      |       |         |
|----------|------|------|--------|------|-------|--------|------|-------|---------|
|          | 0.05 |      |        | 0.25 |       |        | 0.50 |       |         |
| $\alpha$ | min. | max  | avg.   | min. | max   | avg.   | min. | max   | avg.    |
| 0.0      | 31   | 164  | 54.7   | 32   | 348   | 143.7  | 61   | 581   | 207.9   |
| 0.5      | 33   | 271  | 149.0  | 76   | 13668 | 3587.8 | 32   | 57321 | 25123.2 |
| 1.0      | 94   | 2482 | 1182.6 | 197  | 1101  | 490.8  | 95   | 4511  | 1329.8  |
| 1.5      | 84   | 1839 | 717.4  | 31   | 1199  | 359.4  | 31   | 1028  | 390.0   |



Table 5. The computation time in seconds (n=30)

| $\alpha$ | $\beta$ |     |      |      |     |      |      |     |       |
|----------|---------|-----|------|------|-----|------|------|-----|-------|
|          | 0.05    |     |      | 0.25 |     |      | 0.50 |     |       |
|          | min.    | max | avg. | min. | max | avg. | min. | max | avg.  |
| 0.0      | 0       | 2   | 0.9  | 0    | 9   | 1.8  | 0    | 5   | 2.1   |
| 0.5      | 0       | 3   | 1.4  | 1    | 99  | 48.4 | 1    | 834 | 251.0 |
| 1.0      | 2       | 23  | 10.8 | 1    | 8   | 3.6  | 1    | 29  | 9.2   |
| 1.5      | 1       | 15  | 6.2  | 0    | 12  | 3.6  | 0    | 8   | 3.2   |

Table 6. Mean deviation of lower bounds from optimum (n=30)

| $\alpha$ | $\beta$        |                 |                   |                |                 |                  |                |                 |                  |
|----------|----------------|-----------------|-------------------|----------------|-----------------|------------------|----------------|-----------------|------------------|
|          | 0.05           |                 |                   | 0.25           |                 |                  | 0.50           |                 |                  |
| 0.0      | 0.045<br>(797) | 0.064<br>(1484) | 0.058<br>(1193.5) | 0.044<br>(669) | 0.072<br>(1248) | 0.052<br>(914.9) | 0.051<br>(379) | 0.088<br>(735)  | 0.068<br>(581.3) |
| 0.5      | 0.017<br>(533) | 0.052<br>(958)  | 0.034<br>(837.0)  | 0.000<br>(265) | 0.153<br>(512)  | 0.097<br>(358.6) | 0.000<br>(12)  | 0.080<br>(257)  | 0.033<br>(127.3) |
| 1.0      | 0.010<br>(108) | 0.070<br>(879)  | 0.051<br>(524.2)  | 0.000<br>(28)  | 0.110<br>(426)  | 0.089<br>(184.4) | 0.000<br>(11)  | 0.182<br>(340)  | 0.171<br>(135.2) |
| 1.5      | 0.052<br>(524) | 0.112<br>(2030) | 0.07<br>(1199.2)  | 0.049<br>(48)  | 0.167<br>(619)  | 0.113<br>(332.8) | 0.065<br>(335) | 0.098<br>(1624) | 0.076<br>(670.0) |

# LARGE NEIGHBORHOOD FOR A PARALLEL SCHEDULING PROBLEM WITH EARLINESS-TARDINESS PENALTIES AND A COMMON DUE DATE

Y. A. Rios Solis\* and F. Sourd

*Laboratoire d'Informatique de Paris VI, LIP6*

*8 Rue du Capitaine Scott, 75015 Paris*

*France*

{yasmin.rios;francis.sourd}@lip6.fr

**Abstract** This paper addresses the parallel machine scheduling problem where jobs have different earliness-tardiness penalties and a common due date. Since the problem is conjectured NP-hard in the strong sense, we propose a large neighborhood search that can be explored by a pseudo polynomial dynamic program algorithm. Computational experiments are presented and compared to a lower bound in order to verify that their gap is reasonable. They are also compared to simple local search based on moves and swaps between jobs.

**Keywords:** Large neighborhood, scheduling, earliness-tardiness, common due date, parallel machines.

## Introduction

During the last years the *just-in-time* production has received a special interest. This concept reflects the aim to produce as close to the due date as possible in order to avoid storage costs when the production is early and additional costs resulting from a tardy delivery. The problem we are interested in stems from this just-in-time philosophy.

A set of  $n$  jobs  $J = \{1, 2, \dots, n\}$  is to be executed on  $m$  identical parallel machines. Each machine  $j$  ( $j = 1, \dots, m$ ) is available from time 0 and can only execute one job at the time. Each job  $i$  is also available from time 0 and has an execution time  $p_i$  which is not preemptive. We

\*Partial funding provided by CONACyT, the Mexican National Council of Science and Technology

considerer for this research the hypothesis of a common due date  $d_i = d$  ( $\forall i \in J$ ), at which *ideally* all the jobs must be executed. From a practical point of view, a common due date appears when a set of jobs must be produced simultaneously to be assembled in a later production level or when a client orders various goods that are to be delivered at the same moment.

The aim is to find a schedule  $S$  that specifies the completion time  $C_i$  of each job. A job is early when it completes before the common due date. Given  $S$ , the earliness of a job  $i$  is defined as  $E_i = \max\{0, (d - C_i)\}$ . Similarly, when a job ends after the due date then its tardiness is  $T_i = \max\{0, (C_i - d)\}$ . Jobs whose completion time is  $d$  are said *on time*. Each job  $i$  has an earliness penalty  $\alpha_i$  and a tardiness one  $\beta_i$ . The total cost of a schedule  $S$  is

$$f(S) = \sum_{i \in J} (\alpha_i E_i + \beta_i T_i) \quad (1)$$

The common due date is said to be *large* when  $\sum_{i \in J} p_i \leq d_{large}$ . This large due date allows to ignore the constraint that makes the schedule to start at or after time 0 on every machine. In an intuitive way, if the optimal cost cannot decrease when the common due date increases, then the due date is unrestricted or large. On the other hand, the due date is called *restrictive* ( $d_{res}$ ) and the schedule depends strongly on it. In the three field notation the problem we are interested in is written as  $P | d_i = d_{large} | \sum_i (\alpha_i E_i + \beta_i T_i)$ .

Baker and Scudder present a survey on the type of objective function we are interested in (equation 1), [Baker and Scudder, 1990]. The survey of common due date scheduling presented by Gordon et al. points out some complexity and algorithmic results for parallel machines [Gordon et al., 2002]. Lauff and Werner present a survey of the most important algorithms for earliness and tardiness scheduling with a common due date in parallel machines [Lauff and Werner, 2004]. Nevertheless, there are not many results for the problem we are interested in.

The problem without earliness or tardiness penalties is investigated by Hall [Hall, 1986] and Sundaraghavan and Ahmed [Sundararaghavan and Ahmed, 1984]. They generalize Kanet's algorithm ([Kanet, 1981]). Emmons proposes a  $\mathcal{O}(n \log n)$  algorithm for  $Q | d_i = d_{large} | \sum (E_i + T_i)$  and  $Q | d_i = d_{large} | \sum (\alpha E_i + \beta T_i)$ , where  $Q$  means that the parallel machines are uniform (different dependent speeds). He models the problem with a transportation formulation [Emmons, 1987]. Kubiak et al. present a  $\mathcal{O}(n^3)$  algorithm for  $R | d_i = d_{large} | \sum (E_i + T_i)$ , where  $R$  denotes the unrelated parallel machine environment (different independent speeds) [Kubiak et al., 1990].

Sun and Wang consider the problem  $P |d_i = d_{large}| \sum_i w_i (C_i - d)$  where the weight  $w_i$  of each job is proportional to its processing time [Sun and Wang, 2003]. They show that even for  $m = 2$  the problem is NP-hard in the ordinary sense. They formulate a dynamic program algorithm and consider two list scheduling heuristics. Chen and Powell propose a column generation algorithm for  $P |d_i = d_{large}| \sum_i (\alpha_i E_i + \beta_i T_i)$ . Then they solve up to 60 job instances in reasonable time by a branch and bound algorithm [Chen and Powell, 1999].

Earliness-tardiness problems are frequently solved in an exact way by pseudo polynomial algorithms. To the best of our knowledge, this kind of algorithm has not been presented for  $P |d_i = d_{large}| \sum_i (\alpha_i E_i + \beta_i T_i)$ . In fact, some authors have conjectured  $1 |d_i = d_{large}| \sum_i (\alpha_i E_i + \beta_i T_i)$  is NP-hard in the strong sense [Lee et al., 1991, Feldmann and Biskup, 2003]. Thereafter,  $P |d_i = d_{large}| \sum_i (\alpha_i E_i + \beta_i T_i)$  would be also NP-hard in the strong sense.

The manner we choose to tackle the problem  $P |d_i = d_{large}| \sum_i (\alpha_i E_i + \beta_i T_i)$  is by a *neighborhood search algorithm* (also known as local search). This kind of algorithms need an initial feasible solution and they search for a better solution by exploring the “neighborhood” of the solution. This solution becomes the current solution and the local search goes on by exploring this new neighborhood. The algorithm stops when the current neighborhood does not furnish a better value. The current solution is called the “local optimum”.

In an intuitive way, the larger the neighborhood is, the better is the quality of the local optimum. But, the larger the neighborhood is, the longer is needed to explore it. For this reason, a larger neighborhood is in practice interesting only when it can be explored in an efficient way. The article of Ahuja et al. presents a survey of very large scale neighborhoods. It focuses on exponential neighborhoods (according to the initial instance) where the best neighbor can be found in polynomial time [Ahuja et al., 2002].

Congram et al. have introduced the *Dynasearch* search algorithm for the one-machine problem when the objective function is to minimize the sum of the weighted tardiness [Congram et al., 2002]. With this technique, neighbors are obtained by job exchanges. Sourd has studied problem  $1|r_i, setup| \sum_i (\alpha_i E_i + \beta_i T_i)$  with this neighborhood technique ( $r_i$  is the release date of job  $i$ ). He proves that the exploration of the *Dynasearch* neighborhood is an NP-hard problem. Nevertheless, it can be explored by in a pseudo polynomial time [Sourd, 2004].

The single machine problem with a restricted due date,  $1 |d_i = d^{res}| \sum_i (\alpha_i E_i + \beta_i T_i)$ , has been studied by Rios and Sourd

[Rios and Sourd, 2005]. Some experimental results are pointed out in order to compare the quality of the local search solution with other heuristics. This work showed out the good quality of this kind of large neighborhood search.

Our purpose is to explore a large neighborhood for  $P|d_i = d_{large}|\sum_i (\alpha_i E_i + \beta_i T_i)$  based on job exchanges: an early job is inserted late in the same machine or in another; a late job is inserted early in any machine; an early job is transferred early to another machine; and vice-versa, a tardy job is transferred late to another machine. This neighborhood is interesting under the assumption that  $P|d_i = d_{large}|\sum_i (\alpha_i E_i + \beta_i T_i)$  is NP-hard in the strong sense. Thus, there would be no pseudo polynomial algorithm to solve it in an exact way.

We are to propose a pseudo polynomial dynamic program algorithm in order to explore a large neighborhood for  $P|d_i = d_{large}|\sum_i (\alpha_i E_i + \beta_i T_i)$ . Section 1 is about the properties of the optimal schedule. The exponential neighborhood is described in section 2. The dynamic program algorithm is presented in section 3 in order to find the best neighbor. Finally, some computational tests are presented in section 4.

## 1. V-shaped schedule

The class of dominant schedules of  $P|d_i = d_{large}|\sum_i (\alpha_i E_i + \beta_i T_i)$  inherits some well known properties from the one machine problem. Those properties allow reducing the searched space. A schedule that verifies the following properties is a *V-shaped* schedule ([Baker and Scudder, 1990]).

**Property 1** *In an optimal schedule, there is no idle time in between the execution of the jobs.*

In the case where there is some idle time between the execution of two jobs in a machine, it is easy to verify that the total cost of the schedule can be reduced by moving one of the jobs toward the common due date.

**Property 2** *In an optimal schedule, at each machine  $j$  early jobs are ordered according to the non-decreasing ratios  $\alpha_i/p_i$ , and the late jobs are ordered according to non-increasing ratios  $\beta_i/p_i$ .*

The proof is based on exchange arguments and it is similar to the proof for the one machine problem (see [Biskup and Feldmann, 2001, Baker and Scudder, 1990]).

**Property 3** *An optimal schedule exists in which for each machine a job is completed exactly at the due date  $d_{large}$ .*

The reader is referred to the following articles for the proof: [Hoogeveen and van de Velde, 1991, Baker and Scudder, 1990].

Therefore, a V-shaped schedule is determined by  $2m$  job sub sets: the set of the on-time job plus the early jobs to be executed on machine  $j$  denoted by  $S_E^j$ , and the set of tardy jobs of machine  $j$ ,  $S_T^j$  ( $j = 1, \dots, m$ ).

Notice it defines a partition of the job set,  $J = \bigcup_{j=1}^m S_E^j \cup S_T^j$ .

## 2. Neighborhood

Let  $S$  be the feasible initial V-shaped schedule. Let  $V_1(S_E^j, S_T^{j'})$  be the neighborhood which allows to explore the early job set of machine  $j$  with the tardy one of machine  $j'$ . In addition, let  $V_2^E(S_E^j, S_E^{j'})$  and  $V_2^T(S_T^j, S_T^{j'})$  be the neighborhood which explores respectively two early job sets and two tardy job sets. The neighborhood we propose for  $S$  is the following neighborhood union:

$$V(S) = \left( \bigcup_{j,j'} V_1(S_E^j, S_T^{j'}) \right) \cup \left( \bigcup_{j < j'} V_2^E(S_E^j, S_E^{j'}) \cup V_2^T(S_T^j, S_T^{j'}) \right)$$

We first consider neighborhood  $V_2^T$ . Let  $J_T(j, j') = \{i \in S_T^j \cup S_T^{j'}\}$  be the union set of tardy jobs on machine  $j$  and  $j'$ . We remark exploring neighborhood  $V_2^T(S_T^j, S_T^{j'})$  corresponds to solved problem  $P2|| \sum_{i \in J_T(j, j')} \beta_i C_i$  ( $P2$  means two identical parallel machines). It can be done in an exact way by a pseudo polynomial algorithm (this problem is NP-hard but not in the strong sense), [Bruno et al., 1974]. Then, a dynamic program can solve this problem in  $\mathcal{O}\left(n \sum_{i \in J_T(j, j')} p_i\right)$  using the Smith's rule (non decreasing  $p_i/\beta_i$  ratios). In the same manner,  $V_2^E$  corresponds to solve the analogous problem  $P2|| \sum_{i \in J_E(j, j')} \alpha_i S_i$ ,

where  $J_E(j, j') = \{i \in S_E^j \cup S_E^{j'}\}$  and  $S_i$  is the starting time of job  $i$ .

Thereafter, we focus on the exploration of  $V_1(S_E^j, S_T^{j'})$ .

In order to describe neighborhood  $V_1$ , we let  $S_E = S_E^j$  and  $S_T = S_T^{j'}$  since the machines are identical and the function cost only depends on the job completion times. Notice exploring  $V_1(S_E, S_T)$  is equivalent to the exploration of a single machine neighborhood. We introduce some supplementary notations (see figure 1). Let  $n_E = |S_E|$  and  $n_T = |S_T|$ . The first  $n_E$  jobs are renamed  $\mathcal{E}_{n_E}, \dots, \mathcal{E}_2, \mathcal{E}_1$  while the last  $n_T$  jobs are renamed  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_{n_T}$ . The earliness and tardiness penalties and

the duration of  $\mathcal{E}_i$  and  $\mathcal{T}_{i'}$  are respectively noted by  $(\alpha_i^\mathcal{E}, \beta_i^\mathcal{E}, p_i^\mathcal{E})$ , and  $(\alpha_{i'}^\mathcal{T}, \beta_{i'}^\mathcal{T}, p_{i'}^\mathcal{T})$ .

We define now two “Remove-Insert” operators which take into account properties 1, 2 and 3:

- $\text{ET}(\mathcal{E}_i)$  removes the early job  $\mathcal{E}_i$  and inserts it late. Property 2 points out the position job  $\mathcal{E}_i$  must take among late jobs of  $S_T$ . We suppose this position is unique: if some ratios are equal then we break the tie according to the initial index of the task. Thus, we define  $R(i) = i'$  if  $\mathcal{E}_i$  must be placed between jobs  $\mathcal{T}_{i'}$  and  $\mathcal{T}_{i'+1}$ . Moreover,  $R(i) = 0$  if  $\mathcal{E}_i$  must be placed before  $\mathcal{T}_1$  and  $R(i) = n_T$  if  $\mathcal{E}_i$  must be inserted in last position.
- The second operator is the opposite move that inserts among the early jobs a late one. This operator is denoted as  $\text{TE}(\mathcal{T}_i)$  if job  $\mathcal{T}_i$  is the one to be put early. As we have defined  $R(i)$ , we define  $L(i)$  as the rank of  $\mathcal{T}_i$  among the early jobs.

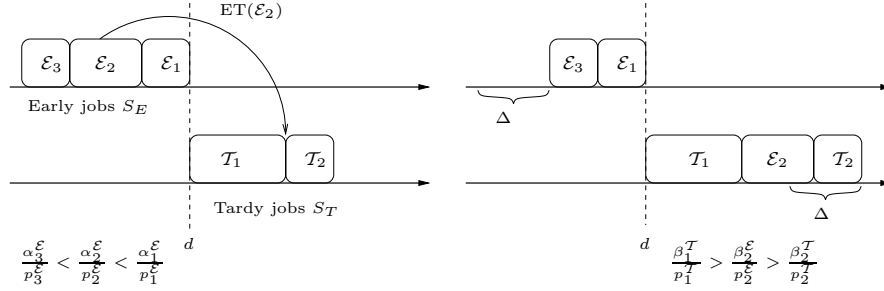


Figure 1.  $\text{ET}(\mathcal{E}_2)$  operation

Figure 1 illustrates the operation  $\text{ET}(\mathcal{E}_2)$  : early job  $\mathcal{E}_2$  is moved tardy.  $R(2) = 1$  since  $\mathcal{E}_2$  must be placed between  $\mathcal{T}_1$  and  $\mathcal{T}_2$ . This move implies a shift  $\Delta = p_2^\mathcal{E}$  from the current schedule on  $S_E$  and  $S_T$ .

The *amplitude* of operator  $\text{ET}(\mathcal{E}_i)$  (resp.  $\text{TE}(\mathcal{T}_i)$ ) corresponds to the pair  $(i, R(i))$  (resp.  $(L(i), i)$ ). Two operations are said *independent* if their amplitudes are nested, i.e., if  $i_1 \geq i_2$  and  $i'_2 \leq i'_1$  for the two amplitudes  $(i_1, i'_1)$  and  $(i_2, i'_2)$ . Neighborhood  $V_1$  is then defined by a combination of ET and TE independent moves.

We have  $2m$  sub sets of early and tardy jobs and we can explore a pair of those sub sets. Making the exploration of a single sub set pair results in a candidate to be the best neighbor. Thereafter,  $S$  has  $m(2m - 1)$  candidate neighbors. We choose the best one, put it as the current solution and reiterate until the end of the local search.

**Lemma 1** *The size of the neighborhood  $V_1$  is  $\Omega(2^n)$ .*

To prove this result it is sufficient to notice the case where  $\alpha_i/p_i \geq \alpha_{i'}/p_{i'}$  implies  $\beta_i/p_i \geq \beta_{i'}/p_{i'}$  for all jobs. Thus, execute independent operations ET and TE between each pair  $\mathcal{E}_i$  and  $\mathcal{T}_i$  is possible.  $\square$

The neighborhood  $V_1$  of  $S$  has an exponential size. Then, by the algorithm we propose in the next section, we are able to find the best schedule of the neighborhood in pseudo polynomial time.

### 3. Exploring the neighborhood $V_1$

In this section we propose a dynamic programming algorithm to explore the neighborhood  $V_1(S_E, S_T)$  for any early job set and any tardy job set. To explore this neighborhood, let  $\delta(i, i', \Delta)$  be the best value of the cost function after executing some combination of independent ET and TE moves between the jobs  $\mathcal{E}_i, \dots, \mathcal{E}_1$  and  $\mathcal{T}_1, \dots, \mathcal{T}_{i'}$  with a shift equal to  $\Delta$ . This last parameter is the shift from the initial schedules  $S_E$  and  $S_T$  and the ones obtained after some operations (see figure 1).  $\Delta$  can take the integer values in the interval

$$I = \left[ - \sum_{i \in S_T} p_i, \sum_{i \in S_E} p_i \right] \quad (2)$$

The recursion for  $0 < i \leq n_E$ ,  $0 < i' \leq n_T$  and  $\Delta \in I$  :

$$\delta(i, i', \Delta) = \min \begin{cases} \delta(i, i' - 1, \Delta) \\ \delta(i - 1, i', \Delta) \\ \delta(i - 1, R(i), \Delta - p_i^{\mathcal{E}}) + CE(i, \Delta) \\ \quad \text{if } R(i) \leq i' \\ \delta(L(i'), i' - 1, \Delta + p_{i'}^{\mathcal{T}}) + CT(i', \Delta) \\ \quad \text{if } L(i') \leq i \end{cases} \quad (3)$$

where  $CE(i, \Delta)$  (resp.  $CT(i', \Delta)$ ) is the cost resulting from removing the early job  $\mathcal{E}_i$  (resp. the tardy job  $\mathcal{T}_j$ ) and placing it in  $S_T$  (resp. in  $S_E$ ).

- First line of the minimum of equation (3) corresponds to leave job  $\mathcal{T}_{i'}$  late, taking into account that eventually some ET and TE moves have been done between jobs  $\mathcal{E}_i, \dots, \mathcal{E}_1$  and  $\mathcal{T}_1, \dots, \mathcal{T}_{i'-1}$ .



- The second line corresponds on leaving job  $\mathcal{E}_i$  early where some ET and TE moves could be done between jobs  $\mathcal{E}_{i-1}, \dots, \mathcal{E}_1$  and  $\mathcal{T}_1, \dots, \mathcal{T}_{i'}$ .
- The third line is about moving job  $\mathcal{E}_i$  tardy among jobs in  $S_T$ . It corresponds to the cost of the scheduling when some operations have eventually been done between jobs  $\mathcal{E}_{i-1}, \dots, \mathcal{E}_1$  and  $\mathcal{T}_1, \dots, \mathcal{T}_{R(i)}$  with a total shift of  $\Delta - p_i^\mathcal{E}$ . Moving  $\mathcal{E}_i$  implies an additional shift of  $p_i^\mathcal{E}$ , this way the total shift is  $\Delta$ . Plus the cost  $CE(i, \Delta)$  composed by the benefice of removing job  $\mathcal{E}_i$  from  $S_E$ , plus the cost of inserting it late in  $S_T$  at the position  $R(i)$  (given by property 2), minus the  $p_i^\mathcal{E}$  shift of early jobs that found themselves before  $\mathcal{E}_i$  plus the  $p_i^\mathcal{E}$  shift of the tardy jobs that are after  $\mathcal{T}_{R(i)}$ . In order to state the cost  $CE(i, \Delta)$ , we note  $C_i^\mathcal{E}$  the completion time of job  $\mathcal{E}_i$  in the original sequence  $S$ . So, the cost of doing operation ET( $\mathcal{E}_i$ ) when the total shift is  $\Delta$ , corresponds to the following formula:

$$CE(i, \Delta) = -\alpha_i^\mathcal{E} (d - (C_i^\mathcal{E} + \Delta - p_i^\mathcal{E})) + \beta_i^\mathcal{E} (C_{R(i)}^\mathcal{T} + \Delta - d) \\ + p_i^\mathcal{E} \left( \sum_{R(i) < h \leq n_T} \beta_h^\mathcal{T} - \sum_{n_E \geq l > i} \alpha_l^\mathcal{E} \right)$$

- The fourth line of equation 3 corresponds to insert tardy job  $\mathcal{T}_{i'}$  into  $S_E$ . Some operations have eventually been done between jobs  $\mathcal{E}_{L(i')}, \dots, \mathcal{E}_1$  and  $\mathcal{T}_1, \dots, \mathcal{T}_{i'-1}$  with a total shift of  $\Delta + p_{i'}^\mathcal{T}$ , resulting on a total shift of  $\Delta$ . Plus the cost  $CT(i', \Delta)$  that results from the gain of removing job  $\mathcal{T}_{i'}$  from  $S_T$ , plus the cost of inserting it early, minus the  $p_{i'}^\mathcal{T}$  shift of jobs that are placed after  $\mathcal{T}_{i'}$  plus the  $p_{i'}^\mathcal{T}$  shift of jobs  $\mathcal{E}_{n_E}, \dots, \mathcal{E}_{L(i')+1}$ . This cost is symmetric to  $CE(i, \Delta)$ .

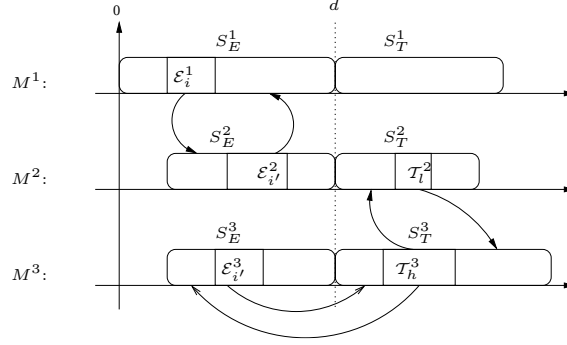
The initialization of the recursion function  $\delta(i, i', \Delta)$  for all  $i = 0, \dots, n_E$  and  $i' = 0, \dots, n_T$  is

$$\delta(i, i', \Delta) = \begin{cases} f(S) & \text{for } \Delta = 0 \\ \infty & \forall \Delta \neq 0 \end{cases}$$

The cost of the best neighbor of  $S_E$  and  $S_T$  is:

$$\min_{\Delta} \delta(n_E, n_T, \Delta)$$

Exploring  $V_1(S_E^j, S_T^k)$  gives the neighbor schedule  $S_{ET}(j, k)$ . In figure 2 this could correspond to explore  $S_E^3$  and  $S_T^3$ . By solving  $V_2^E(S_E^j, S_E^k)$

Figure 2. Exploration of the large neighborhood  $V$ 

we obtain the solution schedule  $S_{EE}(j, k)$ . In figure 2 this could be the exploration of  $S_E^1$  and  $S_E^2$ . And finally, by  $V_2^T(S_T^j, S_T^k)$  we obtain the solution schedule  $S_{TT}(j, k)$  which could correspond to find the best schedule for  $S_T^2$  and  $S_T^3$ . Then, the best neighbor of  $V(S)$  is

$$\min_{j,k} \{S_{ET}(j, k), S_{EE}(j, k), S_{TT}(j, k)\} \quad \forall j, k = 1, \dots, m$$

**Remark 1** When a pair of schedules has been modified, during the next iteration we do not need to compute again the algorithm for the pairs that do not involve those modified schedules.

**Theorem 1** Exploring the large neighborhood of  $S$  takes  $\mathcal{O}\left(m^2 n^3 \sum_{i \in J} p_i\right)$  time.

*Proof.* In recursion (3) of the exponential neighborhood  $V_1$ , computing an operation ET or TE takes at most  $\mathcal{O}(n)$ . Not more than two operations are evaluated by iteration and at most  $(n/2)^2 \cdot (2 \sum_{i \in J} p_i)$  recursive steps are executed. Neighborhoods  $V_2^E$  and  $V_2^T$  can be solved in  $\mathcal{O}(n \sum_i p_i)$ . There are  $m(2m - 1)$  possible pairs to explore, then exploring the large neighborhood takes  $\mathcal{O}\left(m^2 n^3 \sum_{i \in J} p_i\right)$  time.  $\square$

#### 4. Experimental results

The purpose of this section is to point out the quality and the efficiency of the solutions obtained by the large neighborhood search. The instances we consider in this work are the ones Biskup and Feldmann generated in order to apply some heuristic methods

to the restricted problem  $1|d_i = d_{res}|\sum_i (\alpha_i E_i + \beta_i T_i)$ . Those instances can be found in the OR-Library which is a collection of instances for operations research problems (J. E. Beasley web page, <http://www.brunel.ac.uk/depts/ma/research/jeb/info.html>). We only take into account the instances with 10, 20, 50 and 100 jobs (10 instances for each case) and a due date restrictive factor equal to 1 since we consider a large due date.

The dynamic program algorithm we present for the exploration of the large neighborhood was coded in C++, and the tests have been executed on a Pentium IV with a 2.6 GHz processor and 0.98 Go of RAM. After testing some feasible initial schedules, we finally use a simple heuristic for the experimental results. It is based on the idea of positioning early the jobs with a high tardiness penalty compared to their earliness penalty. Based on this criterion, half of the total number of the jobs are positioned early and the other half is putted tardy (verifying property 2). The distribution of the jobs on the machines is uniform. This heuristic has a complexity of  $O(n \log n)$ . From this feasible initial schedule we start the large neighborhood search.

In order to have a comparison point for our computational results, we use the following lower bound: the linear relaxation of the resource constraints proposed by Kedad-Sidhoum, Rios and Sourd [Kedad-Sidhoum et al., 2005]. This lower bound, denoted by  $LB$ , has the advantage that gives quite good results in little time. We also compare our results with the solutions obtained by an effective heuristic consisting on the following basic moves ([Kedad-Sidhoum et al., 2005]):

- job-swap: two jobs are selected on the same or different machine and interchange their positions
- extract and reinsert: a job is chosen and inserted in any another position
- list-crossover: two schedules are selected, say  $S_i$  and  $S_j$  and two positions  $v_i$  and  $v_j$ .  $S_i$  is formed by its first  $v_i$  jobs and followed by the jobs after position  $v_2$  of  $S_2$ . The inverse is done for  $S_2$ .

We call  $1\_HEUR$  the upper bound resulting from a single descent of this heuristic. Similarly, let  $10\_HEUR$  be the bound we obtain by ten descents (ten random initial sequences are improved by the heuristic, and we keep the best value from them). Let  $UB$  be the solution found by our neighborhood search method. Then, the percentage difference between our upper bound  $UB$  and the lower bound  $LB$  is  $100 * |LB - UB| / UB$ , similarly we can compute  $100 * (1\_HEUR - UB) / UB$  and  $100 * (UB - 10\_HEUR) / UB$ .

Table 1. Average percentage difference between the  $UB$  and the  $LB$ 

| <i>Machines</i> | 10 jobs | 20 jobs | 50 jobs | 100 jobs |
|-----------------|---------|---------|---------|----------|
| 2               | 0.29    | 0.38    | 0.11    | 0.04     |
| 4               | 0.90    | 0.77    | 0.18    | 0.15     |
| 6               |         | 0.60    | 0.41    | 0.19     |
| 8               |         | 2.14    | 0.84    | 0.28     |

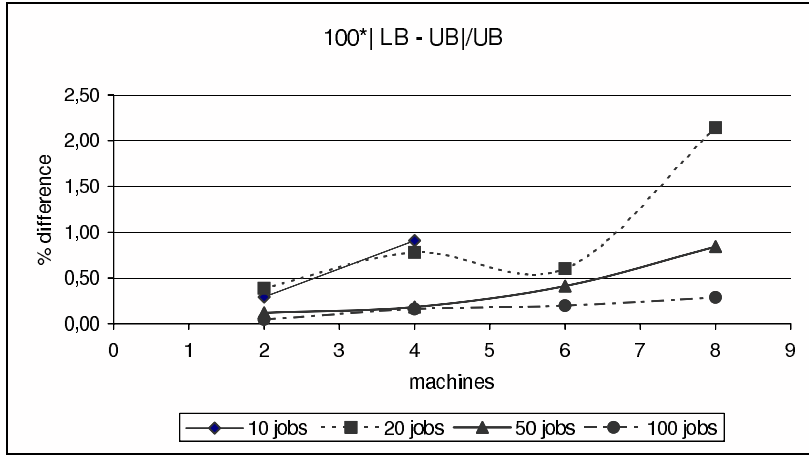
Figure 3. Average percentage differences between  $LB$  and  $UB$ 

Table 1 and figure 3 illustrate the gap between the lower bound  $LB$  and our upper bound  $UB$  for 10, 20, 50 and 100 jobs and for 2, 4, 6 and 8 machines. Notice for 10 job instances we only test 2 and 4 machines. The biggest gap appears for 20 jobs with 8 machines and on the other hand, for the 100 job instances there is no gap bigger than 3%. Since the gaps are reasonable, we can conclude we have good lower and upper bounds. In order to have a better idea of the quality of our bound we compare  $UB$  with  $1\_HEUR$ . Those comparisons are in table 2 and figure 4.

We can observe for the 100 job instances we have in average better bounds than the single descent  $1\_HEUR$  for all the machines cases. For the 50 job instances we obtain better results when the number of machines is 2, 4 or 6. We can notice our large neighborhood search gives better bounds than  $1\_HEUR$  when the number of jobs increases and on the other hand, it gives also good bounds when the number of machines decreases.

Table 2. Average percentage difference between the  $UB$  and  $1\_HEUR$ 

| <i>Machines</i> | 10 jobs | 20 jobs | 50 jobs | 100 jobs |
|-----------------|---------|---------|---------|----------|
| 2               | 0.01    | 0.34    | 0.10    | 0.08     |
| 4               | 0.56    | 0.28    | 0.13    | 0.05     |
| 6               |         | -0.17   | 0.12    | 0.10     |
| 8               |         | -1.59   | 0.01    | 0.29     |

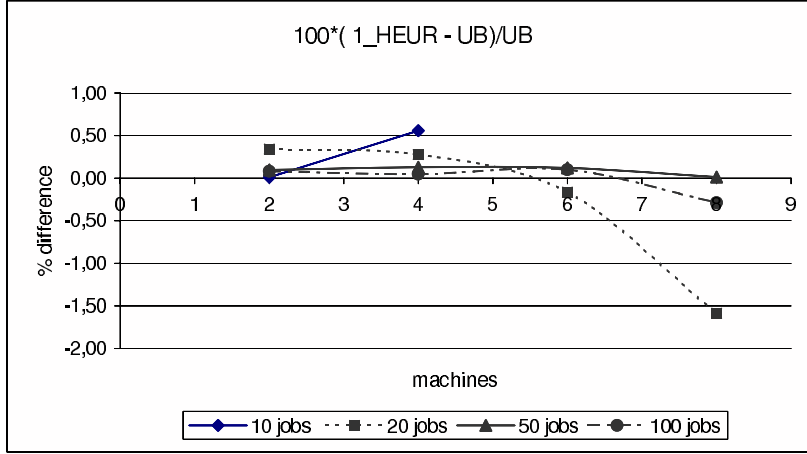
Figure 4. Average percentage differences between  $LB$  and  $1\_HEUR$ 

Table 3 and figure 5 show the average percentage difference between  $UB$  and  $10\_HEUR$ .

Table 3. Average percentage difference between the  $UB$  and the  $10\_HEUR$ 

| <i>Machines</i> | 10 jobs | 20 jobs | 50 jobs | 100 jobs |
|-----------------|---------|---------|---------|----------|
| 2               | -0.15   | -0.16   | 0.01    | 0.02     |
| 4               | -0.75   | -0.67   | -0.01   | -0.02    |
| 6               |         | -0.42   | -0.17   | -0.03    |
| 8               |         | -2.06   | -0.61   | -0.08    |

This time, our bound does not present in average a better quality than  $10\_HEUR$ . Nevertheless, there are few instances where  $UB$  finds a better solution than  $10\_HEUR$ . We notice again that best results are for 100 jobs and for 2 and 4 machines. A diversification of the exploration is

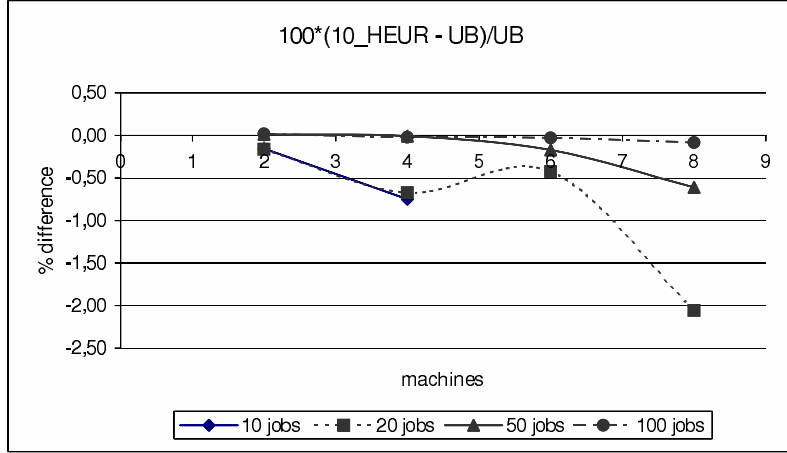


Figure 5. Average percentage differences between  $LB$  and  $10\_HEUR$

interesting since  $10\_HEUR$  obtains in average better results than  $UB$ . Since the large neighborhood search algorithm is time consuming, it would be interesting to have as initial schedule a solution obtained by the heuristic in order to improve the quality of the solution.

The large neighborhood search has a pseudo polynomial complexity time and this is reflected by table 4. The CPU time increases with the number of machines. In fact, the neighborhood search furnish at each iteration  $m(2m - 1)$  different neighbors (see section 2). Figure 6 illustrates the average CPU times for the 100 job instances. The time is quite important but it is no longer than 90 seconds. For 2 machines notice we have better average CPU times than  $10\_HEUR$ .

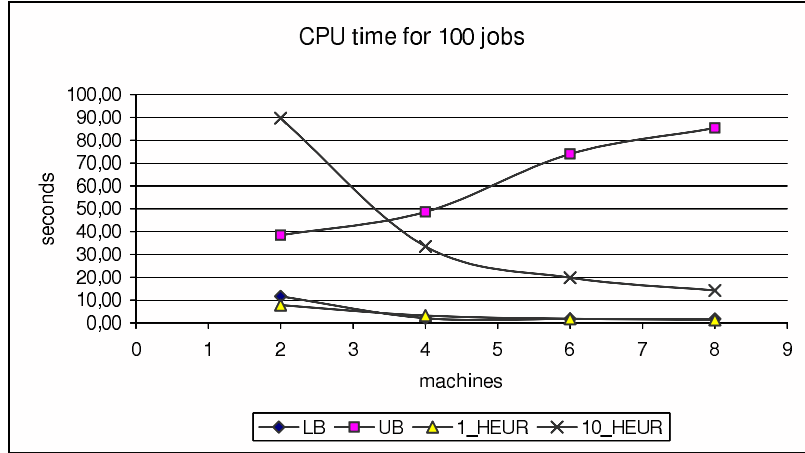
**Remark 2** *Experimental tests have shown interval  $I$  (2) of the pseudo polynomial parameter  $\Delta$  can be reduced without modifying the obtained results but reducing the CPU times. In this paper we have reduced  $I$  by 60%.*

Finally, the maximal number of iterations for the large neighborhood local search are presented on table 5. We remark that the feasible initial solution is quite good and that the local search is ended in no more than 35 iterations.

From those experimental results we can conclude the large neighborhood search furnish good quality solutions when the number of jobs increases and the number of machines decreases. Increasing the number of jobs

Table 4. Average CPU time for *LB*, *UB*, *1\_HEUR* and *10\_HEUR*

|          | Machines | <i>LB</i> | <i>UB</i> | <i>1_HEUR</i> | <i>10_HEUR</i> |
|----------|----------|-----------|-----------|---------------|----------------|
| 10 jobs  | 2        | 0.02      | 0.01      | 0             | 0.02           |
|          | 4        | 0.01      | 0.02      | 0             | 0.02           |
| 20 jobs  | 2        | 0.06      | 0.1       | 0.1           | 0.13           |
|          | 4        | 0.05      | 0.14      | 0             | 0.1            |
|          | 6        | 0.04      | 0.29      | 0             | 0.1            |
|          | 8        | 0.07      | 0.62      | 0.01          | 0.11           |
| 50 jobs  | 2        | 0.9       | 2.61      | 0.52          | 7.2            |
|          | 4        | 0.38      | 5.49      | 0.39          | 3.27           |
|          | 6        | 0.29      | 6.62      | 0.3           | 2.13           |
|          | 8        | 0.28      | 12.79     | 0.12          | 1.37           |
| 100 jobs | 2        | 11.64     | 38.46     | 7.83          | 89.67          |
|          | 4        | 2.06      | 48.62     | 3.2           | 33.55          |
|          | 6        | 1.7       | 74.06     | 1.84          | 19.8           |
|          | 8        | 1.72      | 85.41     | 1.39          | 14.22          |

Figure 6. The *UB* average CPU time for 100 jobs

to be scheduled increases the quality of the solution but the processing time increases also with it.

Table 5. Maximal number of iterations of large neighborhood local search

| <i>Machines</i> | 10 jobs | 20 jobs | 50 jobs | 100 jobs |
|-----------------|---------|---------|---------|----------|
| 2               | 5       | 7       | 9       | 12       |
| 4               | 6       | 11      | 19      | 19       |
| 6               |         | 13      | 23      | 34       |
| 8               |         | 12      | 21      | 35       |

## 5. Conclusions

We have studied a scheduling problem on parallel machines with earliness-tardiness penalties and a large common due date. Since problem  $P |d_i = d_{large}| \sum_i (\alpha_i E_i + \beta_i T_i)$  is conjectured as NP-hard in the strong sense we propose an exponential neighborhood search based on job exchanges between the early schedules and the tardy ones. The exploration of this large neighborhood is done by a dynamic program algorithm in pseudo polynomial time:  $\mathcal{O}(m^2 n^3 \sum_{i \in J} p_i)$ . This way, the best neighbor is identified and becomes the current solution.

The experimental results show out that this large neighborhood search is interesting since the quality of its solutions is satisfactory. As a perspective, we would like to study the restrictive due date problem  $P |d_i = d_{res}| \sum_i (\alpha_i E_i + \beta_i T_i)$  with this neighborhood method.

We have a large neighborhood and an algorithm that are specific for the identical parallel machines. Nevertheless, the generalization for problems where the machines have different independent machine speeds seems not difficult, since the neighborhood and the dynamic program algorithm does not depend on the processing times nor on the machine jobs are executed on. Thereafter, it would be interesting to have some experimental tests in order to verify the efficiency and quality of the solutions.

## References

- [Ahuja et al., 2002] Ahuja, R. K., Ergun, O., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102.
- [Baker and Scudder, 1990] Baker, K. R. and Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38 (1):22–36.
- [Biskup and Feldmann, 2001] Biskup, D. and Feldmann, M. (2001). Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operations Research*, 28:787–801.
- [Bruno et al., 1974] Bruno, J. L., Coffman, E. G., and Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Comm. ACM*, 17:382–387.



- [Chen and Powell, 1999] Chen, Z. L. and Powell, W. B. (1999). A column generation based decomposition algorithm for a parallel machine just in time scheduling problem. *European Journal of operational Research*, 116.
- [Congram et al., 2002] Congram, R. K., Potts, C. N., and van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14:52–67.
- [Emmons, 1987] Emmons, H. (1987). Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics Quarterly*, 34:803–810.
- [Feldmann and Biskup, 2003] Feldmann, M. and Biskup, D. (2003). Single-machine scheduling for minimizing earliness-tardiness penalties by meta-heuristic approaches. *Computers Industrial Engineering*, 44:307–323.
- [Gordon et al., 2002] Gordon, V., Proth, J. M., and Chu, A. C. (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139:1–25.
- [Hall, 1986] Hall, N. G. (1986). Single and multi-processor models for minimizing completion time variance. *Naval Research Logistics Quarterly*, 33:49–54.
- [Hoogeveen and van de Velde, 1991] Hoogeveen, J. A. and van de Velde, S. L. (1991). Scheduling around a small common due date. *European Journal of Operational Research*, 55:237–242.
- [Kanet, 1981] Kanet, J. J. (1981). Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28:643–651.
- [Kedad-Sidhoum et al., 2005] Kedad-Sidhoum, S., Rios, Y. A., and Sourd, F. (2005). Lower bounds for the earliness-tardiness scheduling problem on single and parallel machines. *Revised version*.
- [Kubiak et al., 1990] Kubiak, W., Lou, S., and Sethi, R. (1990). Equivalence of mean flow time problems and mean absolute deviation problems. *Operations Research Letters*, 9:371–374.
- [Lauff and Werner, 2004] Lauff, V. and Werner, F. (2004). Scheduling with common due date, earliness and tardiness penalties for multi-machine problems: A survey. *Mathematical and Computer Modelling*, 40:637–655.
- [Lee et al., 1991] Lee, C. Y., Danusaputro, S. L., and Lin, C. S. (1991). Minimizing weighted number of tardy jobs and weighted earliness-tardiness penalties about a common due date. *Computers and Operations Research*, 18:379–389.
- [Rios and Sourd, 2005] Rios, Y. A. and Sourd, F. (2005). Voisinage exponentiel pour un problème d’ordonnancement avec des pénalités d’avance-retard autour d’une date d’échéance commune. In *Proceedings of ROADEF 2005 (In French)*, pages 133–149.
- [Sourd, 2004] Sourd, F. (2004). Dynasearch neighborhood for the earliness-tardiness scheduling problem with release dates and setup constraints. *Revised version*.
- [Sun and Wang, 2003] Sun, H. and Wang, G. (2003). Parallel machine earliness and tardiness scheduling with proportional weights. *Computers and Operations Research*, 30:801–808.
- [Sundararaghavan and Ahmed, 1984] Sundararaghavan, P. and Ahmed, M. (1984). Minimizing the sum of absolute lateness in single machine and multimachine scheduling. *Naval Research Logistics Quarterly*, 31:325–333.

# **Machine Scheduling**

*Abstracts*

# A DIVIDE-AND-MERGE HEURISTIC FOR SINGLE MACHINE WEIGHTED TARDINESS PROBLEM

Ahmet Keha and John Fowler  
Arizona State University, Tempe, USA

**Abstract:** We introduce a divide-and-merge heuristic for the single machine weighted tardiness problem. Given an instance of the problem, our algorithm divides it into  $k$  (small) subproblem instances using a feasible solution as a basis. After optimizing each subproblem, the heuristic merges their solutions to form a complete solution to the original problem. We also provide some computational results that show the effectiveness of our heuristic.

**Key words:** Heuristic, Single Machine, Total Weighted Tardiness.

## 1. OVERVIEW

In this presentation, we introduce a divide-and-merge heuristic for the single machine minimum total weighted tardiness problem (TWTP). Suppose that we have a single machine scheduling problem with  $n$  jobs. Let  $p_j$ ,  $d_j$ , and  $w_j$  for  $j=1, \dots, n$  be the processing time, the due date and the weight of job  $j$ , respectively. Let  $C_j$  be the completion time of job  $j$ . The tardiness of job  $j$ ,  $T_j$ , is defined as  $\max\{0, C_j - d_j\}$ . Our objective is minimizing the total weighted tardiness, i.e.  $\sum_{j=1}^n w_j T_j$ . TWTP is represented by  $1||\sum_{j=1}^n w_j T_j$  within the scheduling notation of Graham et al (1979).

One approach for solving TWTP is using exact algorithms, i.e. algorithms that are guaranteed to find the optimal solutions. Some examples of exact algorithms are dynamic programming, branch-and-bound, and mixed integer programming (MIP) based algorithms. Abdul-Razaq *et al.* (1990) provides an extensive survey on branch and bound and dynamic programming based algorithms. Since TWTP is NP-Hard (Lenstra *et al.*, 1977) real life size instances of this problem cannot be solved within a reasonable amount of computational time. For example, as discussed in Keha and Fowler (2005), there are 20 jobs instances of TWTP that cannot be solved to optimality by using MIP techniques within an hour with today's computing capabilities. Given the fact that many real world scheduling problems have more than a hundred jobs, we probably will not be able to rely only on exact algorithms to find good solutions.

Since it is not always possible to find the “optimal solution” fast enough, we use heuristics to find “good solutions” faster. Heuristics are usually polynomial time algorithms and a (hopefully) good feasible solution is obtained quickly. Currently, heuristics are the most popular algorithms to solve TWTP.

These two approaches have their own strengths: exact algorithms find the best solution while heuristics find (hopefully) good solutions faster. In this paper we use the divide-and-merge approach that combines the strengths of both approaches. Given an instance of a problem, this approach *divides* the problem into  $k$  (small) subproblem instances using a feasible solution as a basis. After optimizing each subproblem, the approach *merges* their solutions to form a complete solution to the original problem. This approach has recently been introduced by Keha and Fowler (2005).

In this presentation, we introduce the divide-and-merge heuristic. We start with an example application of the approach and then give a formal description of a generic divide-and-merge heuristic. Next, we give results of a set of experiments to test the performance of the divide-and-merge heuristic and to observe the parameters that affect the performance of the approach. We conclude with a discussion of the results and further research issues.

## REFERENCES

- T. S. Abdul-Razaq, C. N. Potts and L. N. Van Wassenhove, ‘A survey of algorithms for the single machine total weighted tardiness scheduling problem’, *Discrete Applied Mathematics*, 26, (1990), 235-253.
- 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*, 5 (1979), 287-326.
- A. Keha, and J.W. Fowler, ‘Divide-and-merge approach for scheduling and other operational level combinatorial optimization problems’, *Department of Industrial Engineering Working Paper Series*, Arizona State University, (2005).
- J.K. Lenstra, A.H.G. Rinnooy Khan, and P. Brucker, Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, 1 (1977) 343-363.

# JOB SHOP SCHEDULING WITH EARLINESS, TARDINESS AND INTERMEDIATE INVENTORY HOLDING COSTS

Kerem Bulbul

*Manufacturing Systems and Industrial Engineering,  
Sabanci University, Istanbul, Turkey*

bulbul@sabanciuniv.edu

**Abstract** Motivated by the problem of scheduling customer orders in a make-to-order supply chain with relatively expensive intermediates, we consider a job shop scheduling problem with the objective of minimizing the sum of tardiness, earliness (finished goods inventory holding) and intermediate (work-in-process) inventory holding costs. Our goal is to develop scheduling heuristics to minimize the total cost and demonstrate the effectiveness of these heuristics computationally.

**Keywords:** machine scheduling; heuristic search; job shop scheduling; shifting bottleneck; earliness; tardiness; work-in-process inventory costs; Dantzig-Wolfe reformulation; column generation

## 1. Introduction

We address the problem of scheduling customer orders in job shops where our main objective is to study the trade-off between inventory holding (both work-in-process and finished goods) and tardiness costs. A study by the World Bank (Roberts, 1999) indicates that inventory holding costs account for 24% of the worldwide logistics costs that exceed \$1 trillion in total. This huge financial impact that makes it essential to reduce inventories, coupled with the requirement for a high-level of on-time delivery performance introduces a trade-off between inventory holding and tardiness costs. Ideally, in order to reduce both work-in-process and finished goods inventory holding (earliness) costs, we would like to delay the production of customer orders as much as possible. Clearly, this approach increases the risk of tardiness, and we need a strategy that manages the trade-off between inventory holding and tardiness costs effectively.

High levels of inventory pose additional challenges in the rapidly changing high-tech industry in which steep price declines over a short period of time are common (Herrick and Verga, 2002), and obsolete inventories are a real danger (Barrett, 2001). In this context, it is clear that goods should not be manufactured far in advance of their delivery times, and that correct scheduling decisions are essential in order to avoid excessive inventory-related costs.

We are particularly interested in settings in which the total earliness cost is not a good indicator of the total inventory holding costs, i.e., when there is significant value added between processing stages. A specific example from the automotive industry is discussed by Bulbul et al., 2004a. In automobile manufacturing, steel is coated to delay corrosion and stored in a compact foil form until it is cut and stamped into different shapes when significantly more space for storage is required. Clearly, the inventory holding costs before and after coating, as well as before and after stamping are different, and these differences must be taken into account in order to make good scheduling decisions. For instance, a scheduling model that only considers the finished goods inventory holding (earliness) cost of a completed automobile body could prescribe a solution that prefers to keep the steel in uncoated condition due to some other constraint in the system. One way of avoiding this situation would be assigning a higher intermediate inventory holding cost

to uncoated steel than that of coated steel. We refer the reader to Bulbul et al., 2004a and Bulbul, 2002 for further examples and a more in-depth discussion.

Motivated partly by the examples and issues described above, we consider a job shop scheduling problem with relatively expensive intermediate products. Our objective is to develop scheduling heuristics to minimize the sum of tardiness, earliness (finished goods inventory holding) and intermediate (work-in-process) inventory holding costs. In the single-machine environment, problems with earliness and tardiness costs have been studied in depth. (See Kanet and Sridharan, 2000 and references therein). Also, in the parallel machine environment many results have been obtained for common due date problems. (See Sundararaghavan and Ahmed, 1984, Kubiak et al., 1990, Federgruen and Mosheiov, 1996, Chen and Powell, 1999a and Chen and Powell, 1999b.) However, relatively little research exists on earliness/tardiness problems in flow- and job shop environments. For instance, Sarper, 1995 and Sung and Min, 2001 consider 2-machine flow shop common due date problems, and Chen and Luh, 1999 develop heuristics for a job shop weighted earliness and tardiness problem. Work-in-process inventory holding costs have been explicitly incorporated in Park and Kim, 2000, Kaskavelis and Caramanis, 1998, and Chang and Liao, 1994 for various flow- and job shop scheduling problems, although these papers consider different objective functions and approaches than ours. Further related references are given by Avci and Storer, 2004 who develop effective local search neighborhoods for a broad class of scheduling problems that includes the job shop weighted earliness and tardiness scheduling problem.

## 2. Problem Description

Consider a non-preemptive job shop with  $m$  machines and  $n$  jobs where  $J_i$  is the set of jobs to be processed on machine  $i$ . The operation sequence of job  $j$  is denoted by  $M_j$  where the  $i^{th}$  operation in  $M_j$  is represented by  $M_{ij}$ ,  $i = 1, \dots, m_j = |M_j|$ . Associated with each job  $j$ ,  $j = 1, \dots, n$ , are several parameters:  $p_{ij}$ , the processing time for job  $j$  on machine  $i$ ;  $r_j$ , the ready time for job  $j$ ;  $d_j$ , the due date for job  $j$ ;  $h_{ij}$ , the holding cost per unit time for job  $j$  while it is waiting in the queue before machine  $i$ ;  $\epsilon_j$ , the earliness cost per unit time if job  $j$  completes its final operation before time  $d_j$ ; and  $\pi_j$ , the tardiness cost per unit time if job  $j$  completes its final operation after time  $d_j$ . All ready times, processing times and due dates are assumed to be integer. For clarity of notation, in the definitions above and the model below, assume that machine  $i$  for job  $j$  corresponds to the machine which performs operation  $M_{ij}$ . However, note that our proposed solution approaches are general and do not rely on such an assumption.

For a given schedule  $S$ , let  $C_{ij}$  be the time at which job  $j$  finishes processing on machine  $i$ , and  $w_{ij}$  be the time job  $j$  spends in the queue before machine  $i$ . The sum of costs for all jobs in schedule  $S$ ,  $C(S)$ , can be expressed as follows:

$$C(S) = \sum_{j=1}^n h_{1j}(C_{1j} - p_{1j} - r_j) + \sum_{j=1}^n \sum_{i=2}^{m_j} h_{ij}w_{ij} + \sum_{j=1}^n \epsilon_j E_j + \pi_j T_j,$$

where  $E_j = \max(0, d_j - C_{m_j,j})$  and  $T_j = \max(0, C_{m_j,j} - d_j)$ . Thus, the  $m$ -machine job shop scheduling problem Jm with earliness, tardiness and intermediate inventory holding costs can be formulated as:

$$\begin{aligned} \min_S \quad & C(S) \\ \text{s.t.} \quad & \end{aligned} \tag{1}$$

$$C_{1j} \geq r_j + p_{1j} \quad \forall j \tag{2}$$

$$C_{i-1j} - C_{ij} + w_{ij} = -p_{ij} \quad i = 2, \dots, m_j, \forall j \tag{3}$$

$$C_{ik} - C_{ij} \geq p_{ik} \text{ or } C_{ij} - C_{ik} \geq p_{ij} \quad \forall i, \forall j, k \in J_i \tag{4}$$

$$C_{m_j,j} + E_j - T_j = d_j \quad \forall j \tag{5}$$

$$w_{ij} \geq 0 \quad i = 2, \dots, m_j, \forall j \tag{6}$$

$$E_j, T_j \geq 0 \quad \forall j \tag{7}$$

The constraints (2) prevent processing of jobs before their respective ready times. Constraints (3), referred to as operation precedence constraints, ensure that any job  $j$  follows its processing sequence  $M_{1j}, \dots, M_{m_j, j}$ . Machine capacity constraints (4) ensure that a machine processes only one job at a time, and a job is finished once started. Constraints (5) relate the completion times of the final operations to the earliness and tardiness values.

Following the three field notation by Graham et al., 1979,  $Jm/r_j/\sum_{j=1}^n \sum_{i=1}^{m_j} h_{ij}w_{ij} + \sum_{j=1}^n (\epsilon_j E_j + \pi_j T_j)$  represents problem Jm. Jm is strongly  $\mathcal{NP}$ -hard because a single-machine special case with all earliness costs equal to zero, i.e., the single-machine weighted tardiness problem  $1/r_j/\sum \pi_j T_j$ , is known to be strongly  $\mathcal{NP}$ -hard (Lenstra et al., 1977).

### 3. Solution Approach

We propose two heuristics in order to solve this problem. Both of these heuristics rely on machine-based decomposition approaches. First, we generalize the Dantzig-Wolfe decomposition and column generation technique that was developed and successfully applied to the  $m$ -machine flow shop version of this problem ( $Fm/r_j/\sum_{j=1}^n \sum_{i=1}^m h_{ij}w_{ij} + \sum_{j=1}^n (\epsilon_j E_j + \pi_j T_j)$ ) by Bulbul et al., 2004a. Second, we propose a shifting bottleneck heuristic for this problem. The shifting bottleneck heuristic was originally developed for the problem  $Jm//C_{max}$  by Adams et al., 1988 and then was applied to job shop scheduling problems in order to minimize the maximum lateness or the makespan under various constraints. In addition, Pinedo and Singer, 1999, Singer, 2001 and Mason et al., 2002 designed shifting bottleneck heuristics for various job shop weighted tardiness scheduling problems.

Observe that Jm would be a linear program (LP) if we knew the sequence of jobs on each machine. Using this property, we often develop two-phase heuristics for earliness/tardiness problems. First, a good job processing sequence is found for each machine, and then idle time is inserted either by a specialized algorithm or by solving a linear program. This latter step is commonly referred to as *timetabling*. For our problem, once jobs are sequenced, and assuming jobs are renumbered in sequence order on each machine, the optimal schedule is found by solving the linear program TTJm below.

$$\begin{aligned} \min_{C_{ij}} \quad & C(S) \\ \text{s.t.} \quad & \end{aligned} \tag{8}$$

$$(2) - (3) \text{ and } (5) - (7)$$

$$C_{ij} - C_{ij-1} \geq p_{ij} \quad \forall i, j \in J_i, j \neq 1 \tag{9}$$

In our proposed solution approaches, we construct job processing sequences either by reformulating Jm using Dantzig-Wolfe decomposition and solving its linear programming relaxation approximately, or by solving the subproblems of a shifting bottleneck heuristic. In both cases, once the job processing sequences are obtained, we find the optimal schedule given these sequences by solving TTJm. The linear program TTJm has  $O(nm)$  variables and constraints and can be solved effectively for typical job shop problem sizes, e.g.,  $m = 10$  and  $n = 10, 15$ , considered in the literature. Furthermore, partial job processing sequences may be accommodated easily in TTJm by specifying only a subset of the constraints (9) that correspond to partial sequences. Thus, TTJm could also be potentially embedded in a branch-and-bound algorithm.

### 4. Work-in-process and Future Research

We are currently working on the development of the heuristics mentioned in the previous section. The column generation algorithm presented by Bulbul et al., 2004a for the flow shop version of our problem exploits the duality between Dantzig-Wolfe decomposition and Lagrangian relaxation in order to solve the linear programming relaxation of the reformulated problem effectively. The single-machine subproblems in the column generation algorithm are either weighted completion time or earliness/tardiness problems with unequal ready times. These subproblems are solved using the approaches developed by Bulbul et al.,

2004b. All of these algorithms can be generalized to the job shop scheduling problem under consideration without much difficulty.

The shifting bottleneck heuristic is an iterative machine-based decomposition algorithm with four basic steps: first, a disjunctive graph representation of the problem is constructed. Second, appropriate single-machine subproblems are formulated and solved for each unscheduled machine. Third, one of the unscheduled machines is selected as the “bottleneck” machine based on some machine criticality measure, and the schedule of this machine is fixed. Finally, the previous scheduling decisions are re-evaluated, and some machines are re-scheduled if necessary. For our problem, the most significant challenge is formulating the single-machine subproblems and designing good heuristics to solve them. Our initial results indicate that these subproblems are generalizations of the single-machine earliness/tardiness scheduling problem  $1/r_j / \sum(\epsilon_j E_j + \pi_j T_j)$  which is strongly  $\mathcal{NP}$ -hard.

Ultimately, we plan to compare these two machine-based decomposition approaches described above in extensive computational tests and hope to obtain insights that could lead to simpler scheduling rules for the job shop scheduling problem with tardiness, earliness and intermediate inventory holding costs.

## References

- Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401.
- Avci, S. and Storer, R. (2004). Compact local search neighborhoods for generalized scheduling. Submitted for publication.
- Barrett, L. (2001). CNET news.com. <http://news.com.com/2100-1033-257278.html>.
- Bulbul, K. (2002). *Just-In-Time Scheduling with Inventory Holding Costs*. PhD thesis, University of California at Berkeley.
- Bulbul, K., Kaminsky, P., and Yano, C. (2001, revised 2004b). Preemption in single machine earliness/tardiness scheduling. Submitted for publication to *Journal of Scheduling*.
- Bulbul, K., Kaminsky, P., and Yano, C. (2004a). Flow shop scheduling with earliness, tardiness and intermediate inventory holding costs. *Naval Research Logistics*, 51(3):407–445.
- Chang, Shi-Chung and Liao, Da-Yin (1994). Scheduling flexible flow shops with no setup effects. *IEEE Transactions on Robotics and Automation*, 10(2):112–122.
- Chen, Haoxun and Luh, P.B. (1999). An alternative framework to Lagrangian relaxation approach for job shop scheduling. In *Proceedings of the 38th IEEE Conference on Decision and Control*, volume 1, pages 913–918.
- Chen, Zhi-Long and Powell, W.B. (1999a). A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operations Research*, 116(1):220–232.
- Chen, Zhi-Long and Powell, W.B. (1999b). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing*, 11(1):78–94.
- Federgruen, A. and Mosheiov, G. (1996). Heuristics for multimachine scheduling problems with earliness and tardiness costs. *Management Science*, 42(11):1544–1555.
- Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.
- Herrick, S.L. and Verga, G. (2002). True inventory costs: A “back of the envelope” method to calculate true inventory savings. <http://www.gvcsolutions.com/Trueinventorycosts.html>.
- Kanet, J.J. and Sridharan, V. (2000). Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, 48(1):99–110.
- Kaskavelis, C.A. and Caramanis, M.C. (1998). Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IEEE Transactions*, 30(11):1085–1097.
- Kubiak, W., Lou, S., and Sethi, S. (1990). Equivalence of mean flow time problems and mean absolute deviation problems. *Operations Research Letters*, 9(6):371–374.
- Lenstra, J.K., Rinnooy Kan, A.H.G., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.
- Mason, S.J., Fowler, J.W., and Carlyle, W.M. (2002). A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, 5(3):247–262.
- Park, Moon-Won and Kim, Yeong-Dae (2000). A branch and bound algorithm for a production scheduling problem in an assembly system under due date constraints. *European Journal of Operations Research*, 123(3):504–518.
- Pinedo, M. and Singer, M. (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46(1):1–17.



Roberts, P.O (1999). Supply chain management: New directions for developing economies.

[http://www.worldbank.org/html/fpd/transport/ports/trf\\_docs/new\\_dir.pdf](http://www.worldbank.org/html/fpd/transport/ports/trf_docs/new_dir.pdf).

Sarper, H. (1995). Minimizing the sum of absolute deviations about a common due date for the two-machine flow shop problem. *Applied Mathematical Modelling*, 19(3):163–161.

Singer, M. (2001). Decomposition methods for large job shops. *Computers and Operations Research*, 28(3):193–207.

Sundararaghavan, P.S. and Ahmed, M.U. (1984). Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. *Naval Research Logistics Quarterly*, 31(2):325–333.

Sung, C.S. and Min, J.I. (2001). Scheduling in a two-machine flowshop with batch processing machine(s) for earliness/tardiness measure under a common due date. *European Journal of Operations Research*, 131(1):95–106.

# SCHEDULING A SINGLE MACHINE TO MINIMIZE A REGULAR OBJECTIVE FUNCTION UNDER SETUP CONSTRAINTS

Philippe Baptiste  
*École Polytechnique, LIX*  
philippe.baptiste@polytechnique.fr

Claude Le Pape  
*ILOG*  
clepape@ilog.fr

**Abstract** Motivated by industrial applications, we study the scheduling situation in which a set of jobs subjected to release dates and deadlines are to be performed on a single machine. The objective is to minimize a regular sum objective function  $\sum_i f_i$  where  $f_i(C_i)$  corresponds to the cost of the completion of job  $J_i$  at time  $C_i$ . On top of this, we also take into account setup times and setup costs between families of jobs as well as the fact that some jobs can be “unperformed” to reduce the load of the machine. We introduce lower bounds and dominance properties for this problem and we describe a Branch and Bound procedure with constraint propagation. Experimental results are reported.

**Keywords:** Machine Scheduling, Constraint Programming

Real manufacturing scheduling problems exhibit a number of difficult features that are often ignored in the literature. Motivated by a new testbed inspired by industrial real life situations [2], we study the one machine scheduling problem with:

- Release dates and deadlines.
- Costs dependent on the completion times of activities.
- Possibilities of leaving some jobs “unperformed.”
- Setup times and costs.

In this problem, a set of jobs  $\{J_1, \dots, J_n\}$  subjected to release dates  $r_i$  and deadlines  $d_i$  are to be performed on a single machine. The pro-

cessing time of  $J_i$  is  $p_i$ . The objective is to minimize a regular (i.e., non-decreasing) sum objective function  $\sum_i f_i$  where  $f_i(C_i)$  corresponds to the cost of completing  $J_i$  at time  $C_i \in [r_i + p_i, d_i]$ . Two extensions of this core problem are considered:

- When the machine is overloaded some jobs can be “unperformed” to reduce the machine load. In such a case, an “unperformance” cost  $u_i$  is associated to each job  $J_i$ . When the job is not scheduled in its time window  $[r_i, d_i]$ , the cost  $u_i$  is added to the objective function. Note that if the  $f_i$  functions are null, the problem reduces to minimizing  $\sum w_i U_i$ , a well-known objective function in scheduling theory (see for instance, [1]).
- Due to manufacturing constraints, *setups* must be performed between jobs with different machine feature requirements. We rely on the following model: there are  $q$  families of jobs and  $\phi(J_i) \in \{1, \dots, q\}$  denotes the family of the job  $J_i$ . Within the same family, there is no transition time nor cost. But between consecutive jobs of families  $\phi_1$  and  $\phi_2$ , at least  $\delta(\phi_1, \phi_2)$  units of time must elapse. Moreover, a cost  $f(\phi_1, \phi_2)$  is associated to this setup.

We describe a new lower bound for this very general problem and a Branch and Bound procedure with constraint propagation. We also introduce a set of dominance rules that reduce the search space.

To our knowledge, this is the first exact procedure for such a general problem. We tested our procedure on 88 instances of the Manufacturing Scheduling Library (MaScLib) [2], available at [www2.ilog.com/masclib](http://www2.ilog.com/masclib). We closed 46 of these instances. Note that when jobs can be unperformed or are subjected to setup constraints, small instances with 24 or 30 jobs are still open. We hope this will encourage other researchers to tackle the problem described in this paper.

Further work includes the development of heuristics to more efficiently explore the search space and the generalization of this procedure to more complex scheduling problems, eg, to a multi-machine environment.

## References

- [1] Peter Brucker. *Scheduling Algorithms*. Springer, 2001.
- [2] W. Nuijten, T. Bousonville, F. Focacci, D. Godard, and C. Le Pape. Towards an industrial manufacturing scheduling problem and test bed. In *Proceedings of PMS'04, Nancy*, 2004.

# EFFICIENT ALGORITHMS FOR MACHINE SCHEDULING PROBLEMS WITH EARLINESS AND TARDINESS PENALTIES

Guang FENG and Hoong Chuin Lau\*

*School of Computing*

*National University of Singapore, 117543*

(fengguan, lauhc)@comp.nus.edu.sg

**Abstract** In this paper, we study multiple machine scheduling problems with early/tardy penalties and sequence dependent setup times. One commonality among works in the literature is that virtually all algorithms decompose the problem into two subproblems - the sequencing problem and the timetabling problem. Sequencing problem focuses on assigning a job to some machine and determine the execution sequence on each machine. We call such assignment a semi-schedule. The timetabling problem focuses on finding an optimal schedule for each semi-schedule. The timetabling problem is polynomial-time solvable. The major challenge of finding good solutions is hence to find good quality semi-schedule, which is strongly NP-hard in general. This paper has two contributions. We first propose a more computationally efficient timetabling algorithm. We then apply the squeaky wheel optimization meta-heuristics to solve the sequencing problem. Finally, we will demonstrate the combined strength of our proposed algorithms.

**Keywords:** Machine scheduling, Meta-heuristic search

## 1. Introduction

Scheduling problems arise commonly in manufacturing and logistics domains. As business move toward greater customer focus, the underlying scheduling problems need to be extended to cater to those needs. Traditionally, only lateness (or tardiness) is a concern. Researchers considered only regular performance measures, which means that the measures are non-decreasing functions of job completion times, such as mean weighted flow times, makespan, mean weighted tardiness. Just-

\*Corresponding Author

in-time management enriched manufacturing and logistics practices and raised a new class of non-regular performance measures to the underlying scheduling problems: the earliness related performance measures. One example is the inventory costs of the final products if the customer does not accept an early shipment. When the scheduling problems consider both earliness penalties and lateness penalties, they are called ET problems.

In this paper, we use the following notations. There are  $N$  jobs to be processed on  $M$  identical machines. A job can be processed on any machine with no machine preferences. Each job is processed exactly once non-preemptively. All jobs are ready at time 0. Job  $i$  has processing time  $p_i$  and due time  $d_i$ . The unit penalty/cost for earliness is  $\alpha_i$ , and that for lateness is  $\beta_i$ . Often, we need to impose setup time  $s_{ij}$  between two successive jobs  $i$  and  $j$ . A schedule for such a problem is defined as the set of completion times  $c_i$  for all jobs. The total penalties of the schedule, or sometimes called total costs, are defined as  $\sum(\alpha_i(0, d_i - c_i)^+ + \beta_i(0, c_i - d_i)^+)$ . For convenience, we define  $E_i = (0, d_i - c_i)^+$  as the earliness of job  $i$ , and  $T_i = (0, c_i - d_i)^+$  as the tardiness of job  $i$ . It is called early if  $E_i > 0$ , or it is called tardy if  $T_i > 0$ . Notice at least one of  $E_i$  and  $T_i$  is zero. When  $E_i = T_i = 0$ , job  $i$  is an on-time job. Sometimes the on-time jobs are also treated as early jobs. The performance measure can hence be defined as  $\sum(\alpha_i E_i + \beta_i T_i)$ . A schedule is optimal if there is no schedule with a lower cost.

MGET refers to the class of Multiple identical machine Generalized ET problems that allow non-uniform sequence dependent setup times, job processing times and due times, and earliness/tardiness penalty weights. The literature on MGET can be divided into two classes: one class formulates MGET by mixed-integer programming models. Two MIP models are Balakrishnan, Kanet, & Sridharan (1998), which solved the general MGET, and Zhu & Heady (2000), which solved MGET with no setup times. These models are not capable to solve problems with more than 12 jobs and/or 3 machines. Hence, a MIP approach is unlikely to be practical for large-scale problems. The other class of literature uses meta-heuristics. Kanet & Sridharan (2000) suggested that it might be useful to decompose the problem into two subproblems: sequencing problem and timetabling problem. The algorithms will be split into two stages: sequencing and timetabling. The sequencing stage generate a semi-schedule, where semi-schedule specifies which machine a job is assigned to and its execution order on that machine. Semi-schedule differs from schedule that semi-schedule does not contain any information on machine execution time. The timetabling stage determines the quality of the semi-schedule(s) by calculating the costs of the semi-schedules. The

results from the timetabling stage will be used by the next sequencing stage to generate new semi-schedules. The two stages repeat iteratively until a satisfying (semi-)schedule is achieved.

The timetabling subproblem for MGET problems (SEQ) is polynomial-time solvable. Gary, Tarjan, & Wilfong (1988); Szwarc & Mukhopadhyay (1995) are the most efficient and widely-used algorithms. Gary, Tarjan, & Wilfong (1988) developed the first polynomial algorithm for SEQ. However, it only presented the SEQ algorithm for the case  $\alpha_j = \beta_j = 1$ , although the authors claimed the general SEQ algorithm can be established in the same way. The time complexity is  $O(N \log N)$ . We shall name this algorithm as SEQ-V1. Szwarc & Mukhopadhyay (1995) developed another version of SEQ algorithm with complexity  $O(NM_C)$ , where  $M_C$  is the number of clusters in the schedule and it is no bigger than  $N$ . We name this algorithm as SEQ-V2. No SEQ algorithm is absolutely better than the other. Other literature developed heuristics algorithms to solve timetabling problems, such as Cheng, Gen, & Tosawa (1995).

Sequencing subproblems, on the other hand, are strongly NP-hard. For large scale problem instances, heuristic approaches are used. Cheng, Gen, & Tosawa (1995) proposed a genetic algorithm to solve MGET with no setup times. The semi-schedule is the chromosome. The chromosome has two kinds of symbols: job id and partition symbol. Each chromosome is a permutation of all jobs with  $M - 1$  partition symbols inserted. These partition symbols divided the permutation into  $M$  subsequences, which were considered as the job sequences on each machine. Radhakrishnan & Ventura (2000) proposed a simulated annealing algorithm for MGET with  $\alpha_i = \beta_i = 1$ , based on job interchange schemes, i.e., selected two jobs and interchange them to generate a neighbourhood solution. This algorithm defined three schemes to perform interchange: “best preceding jobs”, “best succeeding jobs” and adjacent pairs scheme. The best succeeding jobs  $j$  of job  $i$  are the ones with minimum  $|d_i + s_{ij} + p_j - d_j|$ . The best preceding jobs  $j$  of job  $i$  are the ones with minimum  $|d_i - s_{ji} - p_i - d_j|$ . Dong-won Kim et al.(2002) applied simulated annealing to solve MGET where jobs are organized in lots. The setup times only occur between lots, and the jobs are allowed to be assigned to any lot. The neighbourhood schemes including manipulating lots and reassigning job to other lots.

Joslin & Clements (1998); Joslin & Clements (1999) proposed a new meta-heuristic called squeaky wheel optimization (SWO) . They had demonstrated the effectiveness of SWO in optimizing NP-hard problems with regular performance measures, which inspired us to apply it to MGET.

In this paper, we first propose a more computationally efficient SEQ algorithm. Then we propose a SWO algorithm for solving the sequencing problem. We will show the effectiveness by benchmarking our algorithms against the algorithm proposed by Radhakrishnan & Ventura (2000).

## 2. A More Efficient Algorithm for SEQ

SEQ problems are early/tardy scheduling problems for a fixed job sequence. Given a sequence of  $N$  jobs,  $j_1, j_2, \dots, j_N$ , job  $j_i$  must be processed before  $j_{i+1}$ . The jobs have processing times  $p_i$ , due times  $d_i$ , job completion times  $c_i$ , penalties for unit earliness  $\alpha_i$  and penalties for unit tardiness  $\beta_i$ . We do not consider the sequence dependent setup times since such problems can be converted into problems without setup times, according to Davis & Kanet (1993). Since the job sequence is fixed, the SEQ algorithms can be considered as optimal idle-time-insertion algorithms.

Gary, Tarjan, & Wilfong (1988) and Szwarc & Mukhopadhyay (1995) proposed two efficient algorithms, SEQ-V1 and SEQ-V2, with time complexity  $O(N \log N)$  and  $O(NM_C)$  respectively, where  $M_C$  is the number of clusters in the original job sequence. In this paper, we improve their results by proposing an algorithm (SEQ-V3) with time complexity  $O(N \log M_C)$ .

**Definition** Szwarc & Mukhopadhyay (1995). For a job sequence  $N$  jobs ( $j_1, j_2, \dots, j_N$ ), consecutive jobs  $j_i$  and  $j_{i+1}$  are in the same cluster if  $d_{j+1} - d_j < p_{j+1}$ .

**Theorem 1** Szwarc & Mukhopadhyay (1995). Consider any two jobs  $j_i$  and  $j_{i+1}$  in the same cluster. In an optimal schedule, there is no idle time inserted between  $j_i$  and  $j_{i+1}$ .

We describe a cluster by a vector of three variables:  $(c, i, i')$ , where  $c$  is the completion time of the first job of the cluster,  $i$  and  $i'$  are the indexes of the first and last jobs of the cluster respectively.

Figure 1 shows an example of a three-job cluster. The upper part shows the penalty functions of individual jobs with respect to  $c$ . The lower part shows the aggregation of the penalty functions of this cluster with respect to  $c$ . We can observe that the aggregate penalty function of a cluster contains several piecewise-linear line segments. Note that the slope of the lines do not decrease as  $c$  increases. We use symbol  $\Delta^-(c, i, i')$  as the slope of the line left to  $x = c$ , and use symbol  $\Delta^+(c, i, i')$  as the slope of the line right to  $x = c$ .

We define three cluster operations: cluster shift and cluster concatenation and cluster consolidation. Cluster shift needs to update



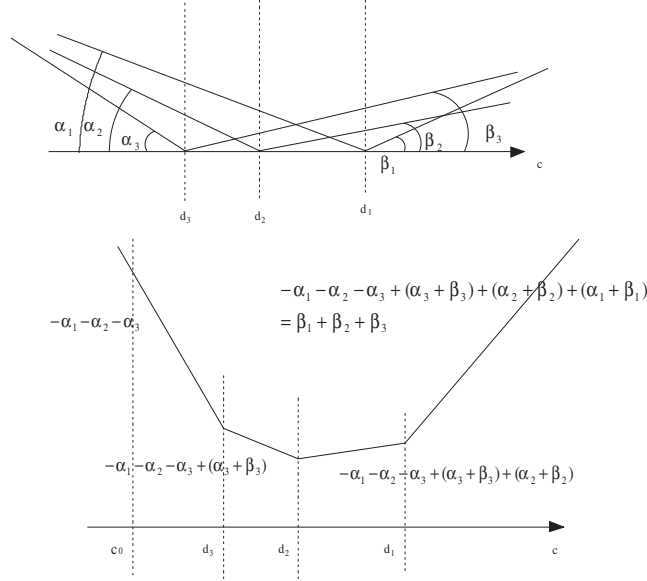


Figure 1. Graph for penalty functions of some cluster

$\Delta^+(d_{l+1}, i, i')$  to  $\Delta^+(d_l, i, i')$  for some  $l, i \leq l \leq i'$ .

$$\Delta^+(d_l, i, i') = \Delta^-(d_l, i, i') + \alpha_l + \beta_l = \Delta^+(d_{l+1}, i, i') + \alpha_l + \beta_l.$$

Suppose cluster  $(c_1, i_1, i'_1)$  and  $(c_2, i_2, i'_2)$  are to be concatenated. Cluster concatenation requires the computation of  $\Delta^+(c, i_1, i'_2)$ . And after this operation, the two clusters will be treated as one and shifted together.

$$\Delta^+(c, i_1, i'_2) = \Delta^+(c_1, i_1, i'_1) + \Delta^+(c_2, i_2, i'_2).$$

Given a cluster  $(c, i, i')$ , cluster consolidation ensures  $\Delta^+(c, i, i') > 0$ . This is achieved by three rules:

- If  $\Delta^+(c, i, i') > 0$ , do nothing.
- If  $\Delta^+(c, i, i') \leq 0$ , and there is some cluster  $(c_1, i_1, i'_1)$ , concatenate cluster  $(c, i, i')$  and  $(c_1, i_1, i'_1)$ . Consolidate the new cluster  $(c_1, i_1, i')$ .
- If  $\Delta^+(c, i, i') \leq 0$ , and there is no cluster before  $(c, i, i')$ , the jobs in this cluster  $(c, i, i')$  are optimal. These jobs are marked exempted from further shifting.

---

**Algorithm 1** SEQ-V3

---

- 1 For all  $1 \leq k \leq N$ , assign  $c_k = \sum_{l=1}^k p_l$ .
  - 2 Split the job sequence into clusters and compute  $\Delta^+(c, i, i')$  and  $W(i, i')$  for all clusters. For cluster  $m$ , define the indexes of the first job and the last job as  $i_m$  and  $i'_m$ .
  - 3 Perform cluster concatenation from the first to the last cluster. Let  $h$  denote the first non-optimal cluster.
  - 4 Since all  $W(i_l, i'_l)$  are pre-sorted, we can merge all  $W(i_l, i'_l)$  ( $h \leq l \leq M_C$ ) into one list  $W$  within  $O(N \log M_C)$  time.
  - 5 If  $h > M_C$ , this algorithm terminates.
  - 6 Find the minimum positive integer  $w \in W$ . If the corresponding job to  $w$  is in some cluster  $h' < h$ , repeat this step.
  - 7 Shift cluster  $(c, i_{h'}, j_{h'})$  from  $c$  to  $w$ .
  - 8 Consolidate cluster  $(c, i_{h'}, j_{h'})$  and update the value of  $h$ . go to step 5.
- 

Now we explain our algorithm SEQ-V3, which is presented in Algorithm 1.

Since each concatenation takes one operation, and there are at most  $M_C$  concatenations, the new scheme can find minimum  $s$  by one operation for each loop, at the expense of  $O(M_C)$  extra operations. The time complexity of all cluster consolidations is  $O(M_C + N) = O(N)$ . Steps 1 – 3 take  $O(N)$  time. Step 4 requires  $O(N \log M_C)$  operations. Steps 5 – 8 take  $O(N)$  operations. Thus, the time complexity of SEQ-V3 is  $O(N \log M_C)$ .

Since our algorithm performs cluster concatenation, we define the concatenated clusters as meta-clusters. We use the symbol  $M(m, n)$  to represent a meta-cluster concatenated from successive clusters

$$(c_{i_m}, i_m, i'_m), (c_{i_{m+1}}, i_{m+1}, i'_{m+1}), \dots, (c_{i_n}, i_n, i'_n),$$

According to SEQ-V2 and its proof, SEQ-V3 is correct if this proposition holds:

**Proposition 1** For any meta-cluster  $M(m, n)$ , for any  $k$ ,  $m \leq k < n$ ,  $\sum_{l=m}^k \Delta^+(c, i_l, i'_l) \geq \sum_{l=m}^n \Delta^+(c, i_l, i'_l)$ .

We prove the proposition by contradiction. Assume there exists some  $K$ , such that

$$\sum_{l=m}^K \Delta^+(c, i_l, i'_l) < \sum_{l=m}^n \Delta^+(c, i_l, i'_l).$$

Then we must have

$$\sum_{l=K+1}^n \Delta^+(c, i_l, i'_l) = \sum_{l=m}^n \Delta^+(c, i_l, i'_l) - \sum_{l=m}^K \Delta^+(c, i_l, i'_l) > 0.$$

Hence, the concatenation of cluster  $K$  will never happen, according to the rules of cluster consolidation, which contradicts our assumption. Thus, the assumption is not possible. The original proposition holds.

### 3. SWO for Sequencing Problems

Sequencing for MGET problems are strongly NP-hard. The knowledge for MGET sequencing is quite limited. For example, sequencing problems of single machine common due date problems have structural property: V-shape property Kovalyov & Kubiak (1999), such that given the earliness/tardiness of the jobs, the sequencing of the jobs is trivial. But for non-common due date problems, there is even no property like that. The knowledge for sequence-dependent setup times is also limited. The limitation in knowledge makes it difficult even to get a good quality heuristics algorithm for MGET problems. In this section, we use a new meta-heuristics (Squeaky Wheel Optimization, or SWO for short) to solve MGET problems.

SWO Joslin & Clements (1998); Joslin & Clements (1999) is a new meta-heuristics for solving NP-hard optimization problems. SWO algorithms operate in two spaces: ordered list space and semi-schedule space. In Joslin & Clements (1998); Joslin & Clements (1999), the ordered list space is called sequence space. Since we have used the term sequence in the discussion of timetabling algorithms, we use the name ordered list instead. The term originally for semi-schedule space is solution space. We feel the semi-schedule is more appropriate because we are using SWO to solve sequencing problems.

SWO algorithms have three major components: constructor, analyser, and prioritizer:

- A constructor takes in an ordered list  $L$ , and generates a semi-schedule  $SS$  according to some greedy algorithm.
- Given an ordered-list  $L$  and the corresponded semi-schedule  $SS$ , the analyser assigns a value to each job. The values are called blames of the jobs.

- The prioritizer adjusts the ordered-list  $L$  according to the blames of the jobs. The new ordered-list  $L'$  will be used as the input to constructor for next iteration.
- This process repeats until a certain quality of semi-schedule is generated.

In SWO, a new semi-schedule is not directly generated from semi-schedules. SWO uses ordered lists as an intermediate control between the generation of two successive semi-schedules. SWO focuses on identifying the jobs that mostly affect the quality of semi-schedules and arrange them at proper positions in the ordered lists. Since we are using greedy algorithms to construct semi-schedules, an earlier position in ordered lists implies certain advantages and/or disadvantages. The ordered lists are controlled in the following way:

- The ordered lists can be considered as a prediction of the importance of the jobs. The position of the job reflects its importance.
- The constructor instantiates the prediction by greedily constructing a semi-schedule. The blames are measurements of the importance of the jobs in semi-schedules. We shall consider the blames as feedbacks to the prediction.
- By comparing the predicted importance and real importance of the jobs, we know whether the prediction is successful. And the next prediction will be adjusted according to the feedbacks.

The methodology can be shown in Figure 2.

### 3.1 Comparison of SWO to Other Meta-Heuristics

Tabu search, genetic algorithms, simulated annealing (SA) and evolutionary algorithms are widely used meta-heuristics. One problem with these approaches is that they do not have a feedback path from the semi-schedule/solution space to the ordered list/sequence space. However, depending on the heuristics algorithms, the ordering of the lists may not be used in some algorithms.

In those heuristics, there are two ways to generate new semi-schedules: direct generation or indirect generation. Direct generation means that a new semi-schedule is generated by modifying/recombining other semi-schedule. A simple but widely used example is adjacent pair interchange. The whole process of direct generation operates in semi-schedule space, which is shown in Figure 3. Genetic algorithms can be

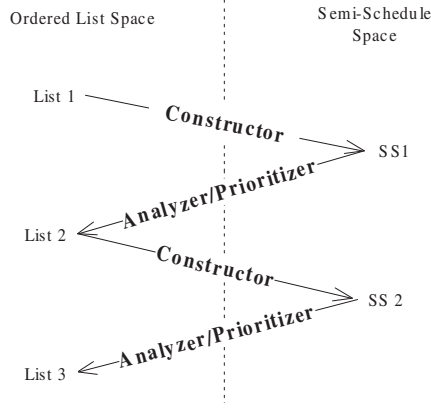


Figure 2. SWO Algorithm Structure

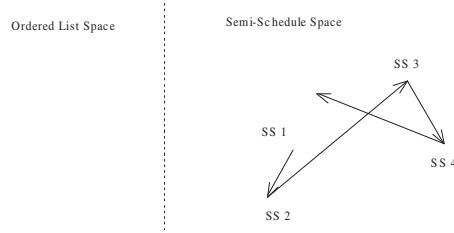


Figure 3. Structure of Direct Generation of New Schedules

another kind of direct generation scheme, depending on implementations. For genetic algorithms, sometimes the chromosome is represented in the form of semi-schedules. But this may not be necessary for all genetic algorithms.

Indirect generation means that a new semi-schedule is not directly generated from semi-schedules. James & Buchanan (1997) is a good example. The list is composed of whether a job is to be scheduled as an early job or a tardy job. The quality of the list is measured by the total costs of the corresponded semi-schedule/schedule. The whole process of generation operates in ordered list space, which is shown in Figure 4. This example is for single machine problems. There is no similar literature for multiple machine problems.

The two generation schemes can be combined to generate new semi-schedules. For example, a problem solved by tabu search algorithm as in James & Buchanan (1997) can be further improved by neighbourhood search, which can be shown in Figure 5.

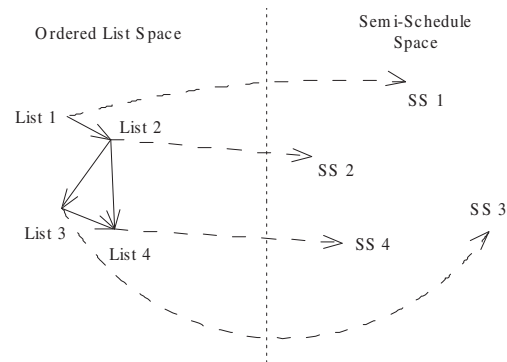


Figure 4. Structure of Indirect Generation of New Schedules

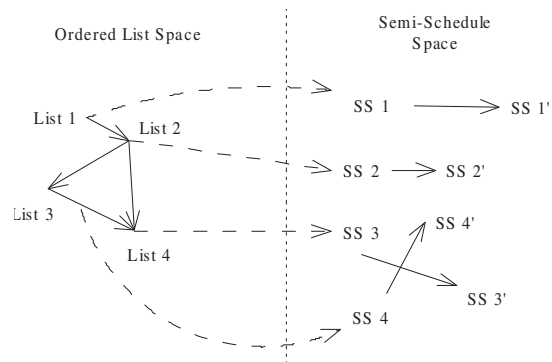


Figure 5. Structure of Combined Generation of New Schedules

In all these three situations, there is no feedback path from semi-schedule space to ordered list space. A possible reason may be that such feedback could not be precise for a difficult problem.

The distinctive feature of SWO is that it tries to utilize the feedback path from semi-schedule space to ordered list space, such that a new list can be generated more effectively. Although the feedbacks are rough, after several iterations, we expect that the quality of semi-schedules can be improved to a certain standard. The quality of the algorithm depends on the design of the constructor, analyser and prioritizer.

#### **4. The Complete MGET-SWO**

We now analyse MGET and discuss some design issues of MGET-SWO. The constructor is a greedy algorithm. The greedy algorithm inserts the jobs into semi-schedules one by one. When inserting each job, we attempt to assign the jobs to each machine; and for each machine, we attempt to insert the job into all possible positions without changing the relative order of jobs already in semi-schedules. For all possible insertions, the best one will be taken to insert the job.

For the greedy algorithm, we consider the jobs' importance according to the sensitivity to the ordering of the list. If the jobs can be put to earlier positions/later positions of the list without changing the total costs by big amount, we consider the jobs insensitive to the ordering of the list. If a job is insensitive, they should be put to an earlier position. If it is sensitive to the ordering of the list, it should be put to a later position.

The analyser uses job reinsertion to determine the sensitiveness to positions. Job reinsertion means that a job is removed from current semi-schedule, and reinserted into the semi-schedule by selecting a position giving lowest total costs with the relative positions of other jobs unchanged. The reduction of total costs brought by job reinsertion is considered as the sensitiveness to ordering of list, or blames in short. Notice that the blames cannot be negative. If the blame is small, we consider the job insensitive to ordering of list. If the blame is large, the sensitiveness to ordering of list is considered large.

The prioritizer sort all jobs in non-decreasing blame order. All sensitive jobs are put to later positions of the next list.

Before presenting the SWO algorithm, we first consider a greedy algorithm (Algorithm 2) that inserts some job to some schedule.

Algorithm 3 shows our overall SWO algorithm.

---

**Algorithm 2 GreedyInsertion(job, schedule)**


---

- 1 For each machine, for each insertion positions, do the following 2 steps
  - 2 Generate the new semi-schedule.
  - 3 Use timetabling algorithm for semi-schedule on each machine.
  - 4 Pick the schedule/semi-schedule with minimal costs as the result, break ties arbitrarily.
- 

---

**Algorithm 3 The Generic SWO Algorithm**


---

- 1 Generate some job permutation L
  - 2 Use a greedy algorithm to construct a semi-schedule SS from L
  - 3 If termination condition is satisfied, STOP
  - 4 Analyse the blames for each job according to SS
  - 5 Prioritizer reorders the jobs according to the blames of the jobs, and generate a new permutation L'
  - 6 Go to step 3.
-



---

**Algorithm 4** Implementation of Analyzer

---

- 1  $n=1$ ; CSEQ is the schedule to be analyzed.
  - 2 While  $n \leq N$  do the following:
  - 3 Remove the  $n^{th}$  job in L from CSEQ
  - 4 CSEQ=GreedyInsertion( $n^{th}$  job in L, CSEQ)
  - 5 Set the blame of  $n^{th}$  job as the cost reduced by the above reinsertion process.
  - 6  $n = n + 1$
- 

We use a simple constructor. During the insertion process, all possible insertion positions are examined, and the one giving the minimum total costs is chosen as the insertion point.

The analyzer is implemented as follows: the blame of each job is defined as the total costs reduced from job reinsertion. SWO-ANA is listed in Algorithm 4. Notice that when a successful reinsertion is found, we will use the improved schedule as the base for future reinsertions, until an even better schedule is found. Essentially, we perform a local search in the analyser algorithm.

## 5. Experimental Results

In this section, we present three sets of experiments to compare the performances of SEQ-V1, SEQ-V2 and SEQ-V3. The test data is generated according to the scheme in Szwarc & Mukhopadhyay (1995).  $p_j$  is generated from uniform distribution in a range  $[1, 100]$ .  $d_j$  is generated from uniform distribution in a range  $[a \times \sum p_j, b \times \sum p_j]$ .  $a - b$  can be one of the six pairs of values:  $0.1 - 0.9$ ,  $0.2 - 0.8$ ,  $0.3 - 0.6$ ,  $0.1 - 1.3$ ,  $0.1 - 1.7$ ,  $0.1 - 2.1$ .  $N$  have four choices: 100, 200, 400 and 500. We use three sets of sequences: increasing, decreasing and random sequences. For increasing sequences, the jobs are sorted in increasing  $d_j$  order. For decreasing sequences, the jobs are sorted in decreasing  $d_j$  order. For random sequences, the jobs are arranged randomly. The time measured is averaged for 5000 runs.

In Table 1, the results are summarized. The numbers are generated by this scheme. For the same problem, we measure the three times:  $t_1$  as the average runtime of SEQ-V1,  $t_2$  as the average runtime of SEQ-V2,  $t_3$  as the average runtime of SEQ-V3. The improvement is calculated as

| N   | a-b     | Increasing | Decreasing | Random  |
|-----|---------|------------|------------|---------|
| 100 | 0.1-0.9 | 13.68 %    | -3.84 %    | 13.37 % |
|     | 0.2-0.8 | 58.29 %    | 8.18 %     | 46.93 % |
|     | 0.3-0.6 | 2.84 %     | 16.11 %    | 15.13 % |
|     | 0.1-1.3 | 26.68 %    | 19.59 %    | 48.00 % |
|     | 0.1-1.7 | 0.00 %     | 26.99 %    | 39.96 % |
|     | 0.1-2.1 | -4.86 %    | 22.16 %    | 41.39 % |
| 200 | 0.1-0.9 | 18.28 %    | 21.84 %    | 18.43 % |
|     | 0.2-0.8 | 16.67 %    | 17.90 %    | 19.07 % |
|     | 0.3-0.6 | 15.28 %    | 16.62 %    | 16.59 % |
|     | 0.1-1.3 | 29.16 %    | 20.77 %    | 41.82 % |
|     | 0.1-1.7 | 10.30 %    | 25.05 %    | 38.17 % |
|     | 0.1-2.1 | 6.44 %     | 28.10 %    | 29.72 % |
| 400 | 0.1-0.9 | 64.22 %    | 25.88 %    | 8.32 %  |
|     | 0.2-0.8 | 61.65 %    | 27.74 %    | 21.05 % |
|     | 0.3-0.6 | 24.27 %    | 25.02 %    | 21.27 % |
|     | 0.1-1.3 | 27.58 %    | 32.44 %    | 44.73 % |
|     | 0.1-1.7 | 22.04 %    | 34.09 %    | 37.61 % |
|     | 0.1-2.1 | 10.48 %    | 36.05 %    | 38.00 % |
| 500 | 0.1-0.9 | 20.86 %    | 32.00 %    | 52.39 % |
|     | 0.2-0.8 | 72.63 %    | 34.85 %    | 15.82 % |
|     | 0.3-0.6 | 23.23 %    | 23.90 %    | 21.33 % |
|     | 0.1-1.3 | 30.82 %    | 31.20 %    | 44.95 % |
|     | 0.1-1.7 | 14.80 %    | 33.54 %    | 39.57 % |
|     | 0.1-2.1 | -0.18 %    | 35.85 %    | 36.35 % |

Table 1. Improvement in runtime of SEQ-V3

follows:

$$100\%(1 - \frac{t_3}{\min\{t_1, t_2\}})$$

, which is the percentage of time savings compared to the best results from SEQ-V1 and SEQ-V2.

The results indicate that for most of the cases, SEQ-V3 is the fastest. And for many of the cases, the runtime is reduced by a big amount (up to 72.63%). There are a few cases that SEQ-V3 is slower than the best performer of SEQ-V1 and SEQ-V2. But in the worst case, SEQ-V3 takes only 4.86% extra time than the best performer, which is not a big difference. Compared to SEQ-V1 and SEQ-V2, SEQ-V3 requires quite a mount of data preprocessing. Thus, in certain cases, SEQ-V3 may not be the fastest. In general SEQ-V3 is a better choice for SEQ problems.

MGET-SA is the simulated annealing algorithm proposed by Radhakrishnan & Ventura (2000). Their algorithm does not perform well. We have made these modifications to their original algorithm, to get a new algorithm MGET-SA2:

- Their adjacent pair interchange algorithms have cycling problems, i.e., the algorithm may repeat a few swaps continuously. We solved this problem by allowing the swap of any two jobs once only, i.e., when applying the adjacent pair interchange algorithm, each swap is recorded such that all future swaps of the same pair of jobs will be disallowed.
- The algorithm's performance can be improved by adding a random pair interchange algorithm as local search.

The algorithm MGET-SWO is the SWO algorithm we have presented.

The set of test problems are generated from the original data generation scheme used by Radhakrishnan & Ventura (2000):  $p_j$  and  $s_{ij}$  are generated from uniform distribution on  $[1, 20]$ .  $d_j$  is generated from uniform distribution on  $[0.225 \times \sum p_j, 0.275 \times \sum p_j]$ .

The results can be summarized in Table 2. Note that the values in the table are the average results of 30 runs. We can see that MGET-SA2 improved MGET-SA, and MGET-SWO performs better than simulated annealing algorithms.

## 6. Conclusion and Future Works

In this paper, we made the contributions in two aspects:

- We distinguished SEQ-V1 and SEQ-V2 as different schemes to manage clusters. We analysed the strengths and weaknesses of

| M  | N   | MGET-SA                          |       | MGET-SA2 |         | SWO     |         |
|----|-----|----------------------------------|-------|----------|---------|---------|---------|
|    |     | Costs                            | Time  | Costs    | Time    | Costs   | Time    |
| 2  | 10  | 189.933                          | 0.76  | 183.767  | 0.988   | 189.433 | 0.0287  |
| 5  | 15  | 114.267                          | 0.264 | 109.2    | 0.248   | 104.967 | 0.0761  |
| 10 | 50  | 455.267                          | 46.56 | 398.1    | 43.88   | 397.733 | 1.276   |
| 15 | 80  | 680.533                          | 104.2 | 525.367  | 95.2061 | 484.367 | 3.839   |
| 13 | 297 | No Improvement. Initial Cost=195 |       |          |         | 96.133  | 406.433 |

Table 2. The Performance Comparison of algorithms for MGET

SEQ-V1 and SEQ-V2, and proposed a more efficient SEQ algorithm that combined the strength of SEQ-V1 and SEQ-V2 that performs fastest both analytically and experimentally.

- We demonstrated the effectiveness of SWO on this problem. While SWO is a promising meta-heuristics for NP-hard optimization problems, we like to point out that applying SWO on other problems may present these challenges (limitations):
  - SWO structure does not imply efficiency, but the possibility of an efficient algorithm. The main difficulty lies in the design of a greedy algorithm and a proper importance recognition scheme, which is not a trivial pursuit.
  - The variance of the solutions can be fairly large. For our algorithms, the standard deviation of total costs is 4% times of the average costs.

Contrary to problems with regular performance measures that have extensive literature, results for ET problems are limited. Most of the works are meta-heuristics based, which lack mathematical rigors. It remains to see if the results for regular performance measures can be extended to ET performance measures.

The current dominating factor of runtime of SEQ algorithms is the sorting of all  $W(i_l, j_l)$ . The current scheme presorts all values in the set  $W = \bigcup W(i_l, j_l)$ . We should notice that the presorting of all numbers is not always necessary. On average, the schedule will become optimal before  $W$  becomes empty, which means that part of the presorting is not necessary. Since we do not know the exact number of values that will be used, we suspect that a smarter scheme can perform sorting only when necessary thereby reducing the time complexity.

## References

- Balakrishnan, N.; Kanet, J. J.; and Sridharan, S. V. 1998. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers and Operations Research* 26:127–141.
- Cheng, R.; Gen, M.; and Tosawa, T. 1995. Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithm. *Computers Industrial Engineering* 29(1-4):513–517.
- Davis, J. S., and Kanet, J. J. 1993. Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics* 40:85–101.
- Dong-won Kim, d.-w. K.; Kim, K.-H.; Jang, W.; and Chen, F. F. 2002. Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing* 18:223–231.
- Gary, M. R.; Tarjan, R. E.; and Wilfong, G. T. 1988. One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research* 13:330–348.
- James, R., and Buchanan, J. 1997. A neighbourhood scheme with a compressed solution space for the early/tardy scheduling problem. *European Journal of Operational Research* 102(3):513–527.
- Joslin, D., and Clements, D. 1998. "squeaky wheel" optimization. In *Proceedings of AAAI98*, 340–346.
- Joslin, D., and Clements, D. 1999. "squeaky wheel" optimization. *Journal of Artificial Intelligence Research* 10:353–373.
- Kanet, J. J., and Sridharan, V. 2000. Scheduling with inserted idle time: Problem taxonomy and literature review. *Operations Research* 48(1):99–110.
- Kovalyov, M. Y., and Kubiak, W. 1999. A fully polynomial approximation scheme for the weighted earliness-tardiness problem. *Operations Research* 47(5):757–761.
- Radhakrishnan, S., and Ventura, J. A. 2000. Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research* 38(10):2233–2252.
- Szwarc, W., and Mukhopadhyay, S. K. 1995. Optimal timing schedules in earliness-tardiness single machine sequencing. *Naval Research Logistics* 42(7):1109–1114.
- Weng, M., and Sedani, M. 2002. Schedule one machine to minimize early/tardy penalty by tabu search. [citeseer.ist.psu.edu/weng02schedule.html](http://citeseer.ist.psu.edu/weng02schedule.html).
- Zhu, Z., and Heady, R. B. 2000. Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. *Computers and Industrial Engineering* 38:297–305.

# STABLE SINGLE-MACHINE SCHEDULES

Roel Leus\* and Willy Herroelen

*Katholieke Universiteit Leuven*

*Leuven, Belgium*

Roel.Leus@econ.kuleuven.ac.be; Willy.Herroelen@econ.kuleuven.ac.be

**Abstract** Robust scheduling aims at the construction of a schedule that is protected against uncertain events. A stable schedule is a robust schedule that changes only little when variations in the input parameters arise. We present a model for single-machine scheduling with stability objective, and we propose a branch-and-bound algorithm for solving an approximative formulation of the model.

**Keywords:** single-machine scheduling, uncertainty, branch-and-bound, stability

## 1. Introduction

Manufacturing schedules regularly suffer disruptions from a variety of sources. Despite these uncertainties, a deterministic ‘baseline’ schedule is usually constructed, which forms the basis for communication with suppliers and subcontractors as well as for commitments to customers [1, 3]. During execution, regular schedule updates will be in order. For coordination purposes, it is desirable that the actual start of each job remain close to its initial baseline starting time. Our goal is to produce a schedule that is protected from the undesirable consequences of rescheduling (and therefore *robust*). A baseline schedule is *stable* if there is little deviation between the baseline and the executed schedule [2].

## 2. Problem statement

A set of jobs  $N$ ,  $|N| = n$ , with deterministic baseline durations  $d_i$  is to be scheduled on a single machine. A schedule specifies a starting time  $s_i$  for each job  $i$ . There is a common deadline  $\omega$  for all the jobs:  $s_i + d_i \leq \omega, \forall i \in N$ . The actual duration of  $i$  is a stochastic variable

\*Postdoctoral Fellow of the Research Foundation - Flanders (FWO - Vlaanderen)

$D_i$ . We assume that job execution does not start before the baseline starting time, which guarantees that actual production will strictly cling to the baseline if no disruptions occur, so  $s_i \leq S_i, \forall i \in N$ , with  $S_i$  the actual starting time of job  $i$ . In case of disruption, we right-shift the remaining jobs without re-sequencing. If we define  $[i]$  to be the job that is scheduled in the  $i$ -th position, then  $S_{[1]} = s_{[1]}$  and  $S_{[i]} = \max\{s_{[i]}; S_{[i-1]} + D_{[i-1]}\}, i = 2, \dots, n$ .

A non-negative integer cost  $c_i$  is incurred per unit-time deviation in  $s_i$  to penalise the resulting system nervousness and shop-coordination difficulties and the delivery delay towards the customer. The expected weighted deviation between *actual* and *planned* job starting times is our stability measure: we minimise  $\sum_N c_j E[S_j - s_j]$ , with  $E[\cdot]$  the expectation operator. Stochastic job duration  $D_i$  is discrete. Specifically, we let random variable  $L_i$  denote the increase in  $d_i$  if  $i$  is disrupted, which takes place with probability  $\pi_i$ . Disruption scenarios for job  $i$  are gathered in set  $\Psi_i$ . Durations for different jobs are independent.

To evaluate the objective-function value, little less can be done in general than to consider all  $\prod_N (|\Psi_i| + 1)$  possible combinations of disruption scenarios. We therefore develop a model that focuses only on the main effects of the separate disruption of each of the  $n$  jobs. The model assumes that exactly one job suffers a disruption from its baseline duration: value

$$p_i = \frac{\pi_i \prod_{\substack{j=1 \\ j \neq i}}^n (1 - \pi_j)}{\sum_{k=1}^n \pi_k \prod_{\substack{j=1 \\ j \neq k}}^n (1 - \pi_j)}$$

is the probability that job  $i$  is the unique disrupted job. The resulting restricted model is useful when disruptions are sparse and spread over time, such that the number of interactions is limited.

In our talk, we present a branch-and-bound algorithm to solve the one-disruption model to optimality. Our computational results show that the output of the model is quite robust to variations in the expected number of disrupted jobs  $\sum_N \pi_i$ .

## References

- [1] Aytug, H., M. Lawley, K. McKay, S. Mohan, R. Uzsoy. 2005. Executing production schedules in the face of uncertainties: a review and some future directions. *Eur. J. Oper. Res.* **161** 86–110.
- [2] Leus, R. 2003. The generation of stable project plans. Complexity and exact algorithms. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium.
- [3] Wu, S., H. Storer, P.-C. Chang. 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. *Comput. Oper. Res.* **20** 1–14.

# LOWER BOUNDS FOR PARALLEL MACHINE SCHEDULING

David Savourey, Antoine Jouglet  
*Université de Technologie de Compiègne*  
{david.savourey, antoine.jouglet}@hds.utc.fr

Philippe Baptiste  
*École Polytechnique, LIX*  
philippe.baptiste@polytechnique.fr

**Abstract** We study the scheduling situation where a set of jobs have to be scheduled on parallel identical machines. Jobs are subjected to release dates and the objective is to minimize a regular min sum criteria  $F = \sum f_i(C_i)$ . In this talk, we review the existing lower bounds for this problem and we introduce two new ones. All of them are theoretically and experimentally compared.

**Keywords:** Machine Scheduling, Theoretical Scheduling

## 1. PROBLEM DEFINITION

We consider the problem where  $n$  jobs have to be scheduled on  $m$  identical parallel machines. Each job  $J_i$  is defined by a release date  $r_i$  and a processing time  $p_i$ . Preemption is not allowed and the objective is to minimize a regular min sum criteria  $F = \sum f_i(C_i)$ .

Although our bounds are valid for any such  $F$ , we focus, as far as experiments are concerned, on some well known criteria such as total completion time ( $\sum C_i$ ), total weighted completion time ( $\sum w_i C_i$ ), total tardiness ( $\sum T_i$ ) and total weighted tardiness ( $\sum w_i T_i$ ). Note that all four corresponding problems  $Pm|r_i|\sum C_i$ ,  $Pm|r_i|\sum w_i C_i$ ,  $Pm|r_i|\sum T_i$  and  $Pm|r_i|\sum w_i T_i$  are *NP*-hard in the strong sense.

## 2. LOWER BOUNDS

We introduce two lower bounds based on the following scheme.



- First we compute a vector  $(C_{[1]}, C_{[2]}, \dots, C_{[n]})$  such that  $\forall i, C_{[i]}$  is a lower bound of the  $i$ th smallest completion time in any schedule.
- Second we define an assignment problem between jobs and the above completion times  $C_{[1]}, C_{[2]}, \dots, C_{[n]}$ . The cost of assigning the job  $J_i$  to the date  $C_{[u]}$  is  $f_i(C_{[u]})$  and we seek to minimize the total assignment cost.

For some objective function, the assignment problem can be solved fast thanks to the problem structure. The general case is solved in cubic time as an assignment problem.

The crucial point of the above mechanism is how to compute the  $C_{[i]}$  values. Our *first lower bound* computes  $C_{[i]}$  as the minimum makespan of any preemptive schedule of at least  $i$  jobs in  $\{J_1, \dots, J_n\}$ . Relying on Horn's formulation [1] for  $P|r_j, pmtn|C_{max}$ , we use a set of "small" MIPs to compute such  $C_{[i]}$  values. For this reason, this lower bound requires a significant amount of time to be computed.

The *second lower bound*, is weaker than the first one since  $C_{[i]}$  is computed as a lower bound of the minimum makespan of any preemptive schedule of at least  $i$  jobs in  $\{J_1, \dots, J_n\}$ . Although this bound is theoretically dominated by the first one, it can be computed fast. Surprisingly, we can also use it to speed up the computation of the the first lower bound.

### 3. CONCLUSION

These lower bounds are compared to an LP relaxation of a time indexed MIP formulation of the problem as well as to more specific bounds, i.e., to bounds that have been proposed for one of the specific problems  $Pm|r_i|\sum C_i$ ,  $Pm|r_i|\sum w_i C_i$ ,  $Pm|r_i|\sum T_i$  and  $Pm|r_i|\sum w_i T_i$  (see for instance [2, 3]).

We have also integrated these bounds in a tree-search procedure. Preliminary results show that they allow us to drastically reduce the search space. Detailed results will be presented at the time of the conference.

### References

- [1] W.A. Horn, *Some simple scheduling algorithms*, Naval Research Logistics Quarterly, 21:177–185, (2004).
- [2] R. Nessah, F. Yalaoui and C. Chu, *Nouvelle borne inférieure pour  $Pm/Ri/SumCi$ : méthode NYCM*, MOSIM'04, Nantes, France (2004).
- [3] F. Yalaoui, C. Chu *New exact method to solve the  $P_m|r_j|\sum C_j$  problem*, to appear in International Journal of Production Economics, 2004.

# Job Shop Scheduling Problem with Multi-purpose Machines and availability constraints

Nozha Zribi   Abdelkader EL Kamel   Pierre Borne  
*LAGIS - Ecole Centrale de Lille*

January 21, 2005

**Abstract.** In this paper, we deal with the Job shop scheduling problem with Multi-Purpose Machine and Availability Constraints. First, we propose a polynomial algorithm solving exactly the problem for two jobs only. Then a lower bound as well as a heuristic approach based on this algorithm are proposed for the general problem.

**Keywords:** job-shop, flexibility, geometric approach, complexity

## 1. Introduction

In the scheduling literature it is generally assumed that machines are available during the whole planning horizon. However, in many real situations, machines may be non-available for processing jobs after a breakdown or during a preventive maintenance activity. In fact, in manufacturing systems, machines are periodically submitted to maintenance operations. This paper deals with the MPM job-shop with limited machine availability. We consider the deterministic model where the unavailability periods corresponding to maintenance tasks are known in advance. We also assume that preemption of operations is not allowed. More precisely, an operation  $O_{ij}$  of job  $J_i$  on machine  $M_k$  starts only if its execution can be finished before  $M_k$  becomes unavailable. The problem we consider is a generalization of the classical job-shop problem and the multi-purpose machine problem studied in (Jurisch, 1992), where machines are available all time.

As compared to the literature dedicated to classical scheduling problems, studies dealing with limited machine scheduling problems are rather rare. Availability constraints have been firstly introduced for a single machine (Adiri et al., 1986) and parallel machines (Schmidt, 1988), (Schmidt, 1988). Lee extensively investigated flow-shop scheduling problems with two machines (Lee, 1996), (Lee, 1997), (Lee, 1999). In particular, the author defined the resumable, non-resumable and semi-resumable models. An operation is called resumable if it can be interrupted by an unavailability period and completed without penalty as soon as the machine becomes available again. If the part of the operation that has been processed before the unavailability period must be partially (respectively fully) re-executed, then the operation is called

semi-resumable (respectively. non-resumable). Recently, flow-shop scheduling problems with two machines and resumable jobs were treated in (Blazewicz et al., 2001), and (Kubiak et al., 2002). Job-shop problem under unavailability constraints has also been considered recently. In (Aggoune, 2002) the author proposed an extension of the geometric approach to solve the two-job shop scheduling problem with availability constraints as well as a branch and bound algorithm with lower bound based on two-job for the job-shop problem with heads and tails and unavailability periods. However to our knowledge MPM Job shop with availability constraints has not been considered yet. The problem is strongly NP-hard since problem without unavailability periods is already strongly NP-hard (Jurisch, 1992). Therefore we propose in this paper an approximation method to solve this problem.

The remainder of this paper is organized as follows. After a description of the considered problem in the following section, we focus, in section 3, on the MPM job shop scheduling problem with two jobs. A polynomial algorithm, which is an extension of the temporized geometric approach introduced by (Aggoune, 2002), is presented. In Section 4 we propose a lower bound for the general problem as well as an heuristic solution.

## 2. problem description

The MPM job-shop with availability constraints may be formulated as follows. There are  $n$  jobs  $J_1, \dots, J_n$  to be processed on a set of  $m$  machines  $\mathcal{R} = (M_1, \dots, M_m)$ . Each machine  $M_k$  can process at most one job at a time. Each job  $J_i$  consists of a sequence of  $n_i$  operations, that must be accomplished according to its manufacturing process. Each operation  $O_{ij}$  ( $i = 1, \dots, n; j = 1, \dots, n_i$ ) can be performed by any machine  $r$  in a given set  $\mu_{ik} \subset \mathcal{R}$  for  $p_{ij}$  time units. The operation is non-preemptive, i.e. it must be accomplished without interruption. Moreover, we assume that machine  $M_k$  is unavailable during giving periods corresponding to preventive maintenance. The starting times and durations of these tasks are known and fixed in advance. The objective is to construct a schedule with a minimum makespan.

According to the terminology concerning the machine availability introduced in (Schmidt, 2000), the studied problem can be denoted by  $J(MPM), NCwin \mid Cmax$ , where NCwin means that non-availability periods are arbitrarily distributed on machines.

The scheduling problem in  $J(MPM)NCwin \mid C_{max}$  can be decomposed in two subproblems:

- a routing subproblem that consists in assigning operations to machines;
- an operation scheduling subproblem associated with each machine aiming to minimize the makespan. This is a Job-Shop scheduling Problem with Availability Constraints  $J(MPM), NCwin \mid C_{max}$ .

### 3. A Geometric representation for

$$J(MPM)NCwin \mid n = 2 \mid C_{max}$$

We focus in this section on the MPM job shop scheduling problem with availability constraints in the particular case of two jobs to be scheduled. We first give a description of the Temporized Geometric Approach (TGA) which is a polynomial algorithm for the two-job shop scheduling problem with limited machine availability. It has been introduced by (Aggoune, 2002) and is an extension of the classical geometric approach developed for the two-job scheduling problem. Then we present a generalization of this approach, which allows dealing with limited machine availability and flexibility of machines.

#### 3.1. TEMPORIZED GEOMETRIC APPROACH FOR CLASSICAL JOB SHOP WITH AVAILABILITY CONSTRAINTS

The geometric approach has been firstly introduced in (Akers and Friedman, 1955). It consists in reducing the two-job shop scheduling problem in the search of a shortest path and thus gives a polynomial algorithm to solve the problem.

In this method the problem can be represented by a two dimension plane with obstacles, defined as follows:

- Each axis representing one job  $J_i = O_{i1}, O_{i2}, \dots, O_{in_i}$  is decomposed into  $n_i$  sub-intervals. The length  $L_{ij}$  of subinterval  $I_{ij}$  is proportional to the processing time  $p_{ij}$  of operation  $O_{ij}$ ,
- Intervals  $O_{1j}$  and  $O_{2k}$  form an obstacle if  $O_{1j}$  and  $O_{2k}$  share the same machine (see Fig. 1),
- the rectangle defined by the origin point O and the final point F, which corresponds to the completion of the two jobs, is the final obstacle.

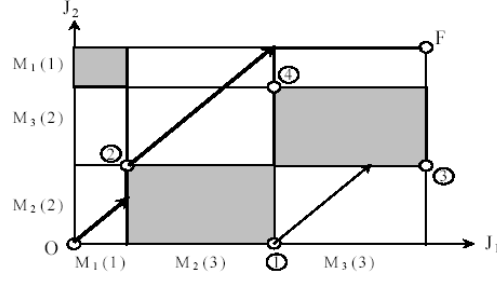


Figure 1. A shortest path in the plane

A feasible solution of the scheduling problem is then a path going from the origin  $O$  to the point  $F$ . Such a path consists of horizontal, vertical and diagonal legs. A horizontal (resp. vertical) leg represents the exclusive progression of job  $J_1$  (resp.  $J_2$ ), whereas diagonal legs correspond to simultaneous executions of the two jobs. Moreover, any path must avoid the interior of the obstacles. This is due to the fact that two operations can not be executed simultaneously on the same machine and are not preemptable.

The TGA is an extension of the geometric approach which exactly solve the problem  $J, N_{Cwin} | n = 2 | Cmax$ . It allows to integrate the evolution of time and so the availability of the machines, based on the definition and the introduction of new vertices, as well as a new and dynamic way to progress from one vertex to its successors.

#### Vertices Characterization and Definitions:

In the classical geometric approach, vertices of the network are the north-west (NW) and south-east (SE) corners of the obstacles hit when going diagonally in the plane. These corners are located at the extremities of the intervals corresponding to operations in conflict. Each vertex can then be defined thanks to its coordinates in the plane: the x-coordinate (resp. y-coordinate) of the vertex corresponds to operation of job  $J_1$  (resp.  $J_2$ ) to be executed. In TGA, some vertices can be located between two lines bounding an operation that is to say inside intervals. For each coordinate of the vertices, an additional information related to the duration already processed of the associated operation is added (Fig. 2). An earliest starting time  $h(S)$  is associated to each vertex  $S$ .  $h(S)$  is the length of the shortest path from the origin to  $S$ .

The set of vertices  $V$  of the network constructed by TGA is composed by the three following types of vertices:

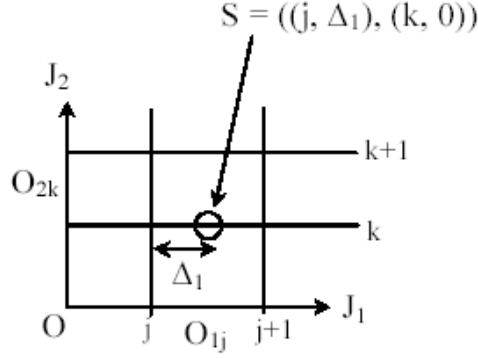


Figure 2. Vertex characterization

- Regular vertices, located at the intersection of horizontal and vertical lines, to which NW and SE corners of obstacles belong.
- Singular vertices, located on a horizontal (resp. vertical) line, which means that the execution of the operation of job  $J_2$  (resp.  $J_1$ ) has not started yet
- Waiting vertices, also located at the intersection of two lines, for which the execution of operations of jobs  $J_1$  and  $J_2$  has not started yet.

A singular vertex is created if the progression of only one job is possible (availability problem for the other job), whereas waiting vertices are created if the progression is possible for none of the two jobs. A waiting vertex is always a duplication of the regular vertex having the same geometric coordinates but not the same earliest starting time. The progression works as follow:

- If the operations of the two jobs cannot start at time  $h(S)$ , the earliest starting time of vertex  $S$ , a waiting vertex is then created.
- If there is an availability problem in the direction  $J_1$  (resp.  $J_2$ ), the progression is made along the vertical (resp. horizontal) line, that corresponds to the execution of the operations of job  $J_2$  (resp.  $J_1$ ) only, until job  $J_1$  (resp.  $J_2$ ) becomes available. A singular vertex, from which a diagonal progression is possible, is added as successor of  $S$ .
- If there is no availability problem, that is to say if the operations of the two jobs can be executed at time  $h(S)$ , the progression works as in the classical geometric approach.

### 3.2. AN EXTENDED APPROACH

We propose a generalization of TGA in order to deal with the flexibility property of the  $J(MPM), NCwin \mid n = 2 \mid C_{max}$ . Like the job shop problem, the scheduling of MPM job shop can be represented in the 2-dimensional plane with potential obstacles that depend on the assignment of machines for the two jobs. Let us define the vertices of the network, the successors of each vertex and the distance between any two vertices.

We develop the algorithm *SuccVertex* allowing to find the successors of each vertex  $S = ((k_1, \Delta_{k_1}^1), (k_2, \Delta_{k_2}^2))$ .

The algorithm progresses diagonally as long as it exists a couple of flexibility machines available for the operations of the two jobs. The corners *SE* and *NW* of the potential obstacles that could be reached from  $S$  are added as successors of  $S$ . If an availability problem occurs in one of the two direction the algorithm *SuccVertexAvailability* is used to define the successors of  $S$  in this case.

In fact the algorithm *SuccVertexAvailability* progresses horizontally or vertically until the progression in the two direction becomes possible, in this case a singular or a regular vertex is added as successor to  $S$ ; another availability problem occurs, and in this case a waiting vertex is added as successor of  $S$  or an unavoidable obstacle is hit.

The algorithm *SuccVertex* is in three steps.

Step 1 is an initialization step. Set  $Ph$  (resp  $Pv$ ) is defined to keep the machines of job  $J_2$  (resp  $J_1$ ) allowing to progress. Set  $R$  is defined to keep the machines allowing to progress until meeting an horizontal and \ or a vertical. This set is used to progress in the next iteration of the program.

$E_1$  is the set of machine flexibility of  $O_{1k_1}$  and  $E_2$  is the set of machine flexibility of  $O_{2k_2}$ . The initialization of these two sets depends on the type of the vertex, we have the following three cases:

- if  $S$  is a waiting vertex,  $E_1$  and  $E_2$  are made of machines used to create this vertex.
- if  $S$  is a singular vertex,  $E_1$  and  $E_2$  are made of machines used to create this vertex.
- if  $S$  is a regular vertex,  $E_1 = \mu_{1v}$  and  $E_2 = \mu_{2h}$ , this means that all machines for  $O_{1v}$  et  $O_{2h}$  could be used for the progression.

In step 2, first we check the availability of the two machines for  $O_{1v}$  et  $O_{2h}$ . If these machines are available, we progress diagonally until a vertical and or an horizontal is met. In step3, we update the current time

$$J(MPM), NCwin \mid C_{max}$$

*current\_time*, the sets  $E_1$ ,  $E_2$ ,  $v$  and  $h$ . The algorithm *SuccVertex* is stopped if the final obstacle is hit or when the diagonal progression is not possible because of availability problems or unavoidable obstacle.



*AlgorithmSuccVertex*


---

**step1**  $h = k_1, \Delta^1 = \Delta_{k_1}^1, v = k_2, \Delta^2 = \Delta_{k_2}^2, \text{current\_time} = h(v)$ ,  
 $E_1$  is the set of machine flexibility of  $O_{1v}$   
 $E_2$  is the set of machine flexibility of  $O_{2h}$   
 $P_v = \emptyset, P_h = \emptyset, R = \emptyset$

**step2**  
**if**  $(h = n1 + 1)$  or  $(v = n2 + 1)$  **then** add  $F$  as a successor. Stop  
**else**  
**for** all machine flexibility  $t_1 \in E_1$  **do**  
    **for** all machine flexibility  $t_2 \in E_2$  **do**  
        calculate the beginning  $s_{\text{current\_time}}^{t_1}, s_{\text{current\_time}}^{t_2}$  and the end  
         $t_{\text{current\_time}}^{t_1}, t_{\text{current\_time}}^{t_2}$  of the unavailability period of  $t_1$  and  $t_2$   
        according to  $\text{current\_time}$   
        check the availability of  $t_1$  and  $t_2$   
        set  $\text{dir1} = 1$  if  $t_1$  is available otherwise  $\text{dir1} = 0$   
        set  $\text{dir2} = 1$  if  $t_2$  is available otherwise  $\text{dir2} = 0$   
        if  $(t_1 = t_2)$  add corners SE and NW of potential obstacle as successors  
    **else**  
        **if**  $\text{dir1} = 1$  and  $\text{dir2} = 1$  **then**  
            **if**  $(p_{1v} - \Delta^1 < p_{2h} - \Delta^2)$  **then**  
                 $\text{progress\_vertic} = 1$   
                 $P_v = P_v \cup t_1$   
                 $R = R \cup t_2$   
            **if**  $(p_{1v} - \Delta^1 > p_{2h} - \Delta^2)$  **then**  
                 $\text{progress\_horiz} = 1$   
                 $P_h = P_h \cup t_2$   
                 $R = R \cup t_1$   
            **if**  $(p_{1v} - \Delta^1 = p_{2h} - \Delta^2)$  **then**  
                 $\text{progress\_diag} = 1$   
                 $P_v = P_v \cup t_1$   
                 $P_h = P_h \cup t_2$   
        **if**  $\text{dir1} = 1$  and  $\text{dir2} = 0$  **then**  
             $\text{progress\_vertic\_availability} = 1$   
             $P_v\text{-availability} = P_v\text{-availability} \cup t_1$   
             $R_h\text{-availability} = R_h\text{-availability} \cup t_2$   
        **if**  $\text{dir1} = 0$  and  $\text{dir2} = 1$  **then**  
             $\text{progress\_horiz\_availability} = 1$   
             $P_h\text{-availability} = P_h\text{-availability} \cup t_2$   
             $R_v\text{-availability} = R_v\text{-availability} \cup t_1$   
        **if**  $\text{dir1} = 0$  and  $\text{dir2} = 0$  **then**  
            add the waiting place  $(h, \Delta^1)(v, \Delta^2)$  as a successor,  
            the earliest starting time of this vertex is fixed to the minimum between  $T_{\text{current\_time}}^{t_1}$   
            and  $T_{\text{current\_time}}^{t_2}$

**step 3**

**if**(*progress\_ vertic\_ availability* = 1)

**SuccVertexAvailabilty** is called

with  $E_1 = P_v$ - *availavility*,  $E_2 = R_h$ - *availability*, *current\_ time*,  $h$ ,  $v$

**if**(*progress\_ horiz\_ availability* = 1)

**SuccVertexAvailabilty** is called

$E_1 = R_v$ - *availability*,  $E_2 = P_h$ - *availavility*, *current\_ time*,  $h$ ,  $v$

**if**(*progress\_ vertic* = 1)

$current\_time = current\_time + p_{1v} - \Delta^1$

$\Delta^2 = \Delta^2 + p_{1v} - \Delta^1$

$v = v + 1$

$\Delta^1 = 0$

$E_1 = \mu_{1v}$

$E_2 = R$

goto step2

**if** (*progress\_ horiz* = 1)

$current\_time = current\_time + p_{2h} - \Delta^2$

$\Delta^1 = \Delta^1 + p_{2h} - \Delta^2$

$h = h + 1$

$\Delta^2 = 0$

$E_2 = \mu_{2h}$

$E_1 = R$

goto step2

**if**(*progress\_ diag* = 1)

$current\_time = current\_time + p_{1v} - \Delta^1$

$h = h + 1$

$\Delta^1 = 0$

$v = v + 1$

$\Delta^2 = 0$

$E_1 = \mu_{1v}$

$E_2 = \mu_{2v}$

goto step2

*AlgorithmSuccVertexAvailability*


---

$h, v, current\_time, E_1, E_2$  are given

**step1**

$\Delta^1 = 0, \Delta^2 = 0, Pv = \emptyset, Ph = \emptyset, R = \emptyset$

**step2**

**if**  $(h = n1 + 1)$  or  $(v = n2 + 1)$  **then** add  $F$  as a successor. Stop

**else**

**for** all machine flexibility  $t_1 \in E_1$  **do**

**for** all machine flexibility  $t_2 \in E_2$  **do**

        calculate the beginning  $s_{current\_time}^{t_1}, s_{current\_time}^{t_2}$  and the end

$t_{current\_time}^{t_1}, t_{current\_time}^{t_2}$  of the unavailability period of  $t_1$  and  $t_2$

        according to  $current\_time$

        check the availability of  $t_1$  and  $t_2$

        set  $dir1 = 1$  if  $t_1$  is available otherwise  $dir1 = 0$

        set  $dir2 = 1$  if  $t_2$  is available otherwise  $dir2 = 0$

**if**  $(t_1 = t_2)$  add corners SE and NW of potential obstacle as successors

**else**

**if**  $dir1 = 1$  and  $dir2 = 0$  **then**

**if**  $current\_time + p_{1v} - \Delta^1 < T_{current\_time}^{t_2}$  **then**

$progress\_vertic = 1$

$P_v = P_v \cup t_1$

$R = R \cup t_2$

**if**  $current\_time + p_{1v} - \Delta^1 > T_{current\_time}^{t_2}$

                progression until  $T_{current\_time}^{t_2}$

                the singular vertex  $(h, \delta^1 + T_{current\_time}^{t_2} - current\_time)(v, 0)$

                is added as successor of S

**if**  $current\_time + p_{1v} - \Delta^1 == T_{current\_time}^{t_2}$

                    progression until  $T_{current\_time}^{t_2}$

                    the regular vertex  $(h + 1, 0)(v, 0)$  is added as successor of S

**if**  $dir1 = 0$  and  $dir2 = 1$  **then**

**if**  $current\_time + p_{2v} - \Delta^2 < T_{current\_time}^{t_1}$  **then**

$progress\_horiz = 1$

$P_h = P_h \cup t_2$

$R = R \cup t_1$

**if**  $current\_time + p_{2v} - \Delta^2 > T_{current\_time}^{t_1}$

            progression until  $T_{current\_time}^{t_1}$

            the singular vertex  $(h, 0)(v, \delta^2 + T_{current\_time}^{t_1} - current\_time)$

            is added as successor of S

**if**  $current\_time + p_{2h} - \Delta^2 == T_{current\_time}^{t_1}$

            progression until  $T_{current\_time}^{t_1}$

            the regular vertex  $(h, 0)(v + 1, 0)$  is added as successor of S

```

if  $dir1 = 0$  and  $dir2 = 0$  then
  add the waiting place  $(h, 0)(v, 0)$  as a successor,
  the earliest starting time of this vertex is fixed to the minimum between  $T_{current\_time}^{t_1}$ 
  and  $T_{current\_time}^{t_2}$ 
step 3
if ( $progress\_vertic = 1$ )
   $current\_time = current\_time + p_{1h} - \Delta^1$ 
   $v = v + 1$ 
   $E_1 = \mu_{1v}$ 
   $E_2 = R$ 
   $\Delta^1 = 0$ 
  goto step2
if ( $progress\_horiz = 1$ )
   $current\_time = current\_time + p_{2v} - \Delta^2$ 
   $h = h + 1$ 
   $\Delta^2 = 0$ 
   $E_2 = \mu_{2h}$ 
   $E_1 = R$ 
  goto step2

```

---

#### Distance Between Two Vertices $S$ and $S'$ :

Let us suppose without loss of generality that  $S$  and  $S'$  are located on horizontal lines, we have:  $d(S, S') = \sum_{k_1}^{k_3-1} p_{1k} - \Delta^1 + \Delta^3$

For other cases, we need to check the availability of machines used to go from  $S$  to  $S'$  in order to calculate the distance between these two vertices, this is not a problem because for any arc going from vertex  $S = ((k_1, \Delta^1), (k_2, \Delta^2))$  to vertex  $S' = ((k_3, \Delta^3), (k_4, \Delta^4))$ , machines of operation  $O_{1k}$  and  $O_{2j}$  are already assigned by algorithm *SuccVertex* for  $k = k_1 \dots k_3$ ,  $j = k_2 \dots k_4$ .

Remarque: If  $F$  is a successor of  $S$  the distance between  $S$  and  $F$  is calculated using only available machines if possible or machines becoming available first. in fact we neglect the other paths.

Note that the distance could be calculated inside the algorithm *SuccVertex*.

**THEOREM 1.** *The set of vertices constructed by applying algorithm *SuccVertex* is sufficient to determine the optimal schedule.*

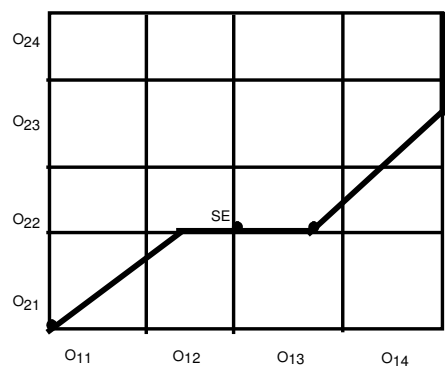


Figure 3. Shortest path

The theorem 1 is due to the fact that TGA gives the optimal schedule in the case of classical job shop (Aggoune, 2002).

### 3.2.1. Complexity Results

**THEOREM 2.** *The scheduling problem  $J(MPM), NCwin \mid n = 2 \mid Cmax$  is polynomial and its complexity is at most equal to  $O(As^4)$  where  $m_1 = K_1 \times n_1, m_2 = K_2 \times n_2$ ,  $K_i = \max(card(\mu_{ij}))$  and  $s = \max(m_1, m_2)$ ,  $A$  is maximum number of unavailability periods that may occur on each machine.*

### 3.2.2. example

Let us consider the following example:

$$J_1 = \{M_3, M_5, 6\}, \{M_3, M_2, 5\}, \{M_1, M_5, 7\}, \{M_3, M_1, 6\}$$

$$J_2 = \{M_3, M_4, 7\}, \{M_2, 4\}, \{M_4, M_2, 6\}, \{M_5, M_3, 4\}$$

$M_1$  is supposed to be unavailable between times 16 and 22,  $M_2$  between times 14 and 17,  $M_3$  between times 8 and 13,  $M_4$  between times 15 and 20,  $M_5$  between times 20 and 25.

The shortest path is obtained in figure 3. It is composed of the following vertices:

$$((0, 0)(0, 0)); ((2, 0), (1, 0)); ((2, 6)(1, 0)); ((3, 0)(3, 0))$$

The optimal makespan is equal to 31, the assignment is as follow:  $O_{11}$  is assigned to  $M_5$ ,  $O_{21}$  to  $M_4$ ,  $O_{12}$  to  $M_2$ ,  $O_{22}$  to  $M_4$ ,  $O_{13}$  to  $M_5$ ,  $O_{23}$  to  $M_4$ ,  $O_{14}$  to  $M_3$ ,  $O_{24}$  to  $M_5$ .

#### 4. The general job shop problem with multi purpose machine and availability constraints

**THEOREM 3.** *A lower bound for  $J(MPM), NCwin \mid C_{max}$  is calculated by considering all  $O(N^2)$  pairs of jobs and taking the maximum solution values obtained by the polynomial algorithm of the previous section*

From the result of the previous section we can deduce a lower bound for the MPM job shop with limited machine availability (theorem 3) as well as a greedy heuristic to calculate a solution for the problem. This heuristic works as follows:

1. The two first jobs are optimally scheduled using the algorithm of the previous section
2. Additional unavailability periods, corresponding to the execution of operations of the two scheduled jobs, are fixed on each machine.
3. The algorithm is applied to the next two jobs of the sequence, taking into account the initial and the new unavailability periods. This procedure continues until all jobs are treated.

If the number of job is odd we need an insertion procedure to schedule the last job. It is based on the following rule: an operation of the last job is inserted in sort that it begins as early as possible, if we have the choice between two machines we choose the machine giving the smallest idle time.

##### 4.1. EXAMPLE

Consider the following example with 5 jobs, 3 operations per job and 5 machines all data created aleatory (Table I).

$M_1$  is supposed to be unavailable between times 4 and 7,  $M_2$  between times 7 and 10,  $M_3$  between times 8 and 9,  $M_4$  between times 5 and 8,  $M_5$  between times 11 and 14.

We report in table II the makespan as well as the value of the lower bound. For this example the value of the lower bound is closed to the obtained makespan.

Note that the heuristic depend on the order of the input sequence. We are actually studying this major point in order to improve the global solution by controlling the initial input sequence.

Table I. Example :  $N = 5$ 

| $J_1$      |          | $J_2$      |          | $J_3$      |          | $J_4$      |          | $J_5$      |          |
|------------|----------|------------|----------|------------|----------|------------|----------|------------|----------|
| $\mu_{1j}$ | $p_{1j}$ | $\mu_{2j}$ | $p_{2j}$ | $\mu_{3j}$ | $p_{3j}$ | $\mu_{4j}$ | $p_{4j}$ | $\mu_{5j}$ | $p_{5j}$ |
| 5          | 3        | 4          | 5        | 2, 3       | 1        | 4          | 2        | 2, 3       | 5        |
| 4          | 4        | 4,3,2      | 2        | 5, 3,4     | 3        | 5          | 9        | 1          | 1        |
| 2,3        | 3        | 4          | 3        | 2, 3       | 4        | 2,3        | 3        | 2, 4,5     | 6        |

Table II. Example :  $N = 5$ 

| makespan | lower bound |
|----------|-------------|
| 26       | 24          |

## 5. Conclusion

We have investigated in this paper MPM job shop scheduling problems under availability constraints. We have proposed an extension of the temporized geometric approach to deal with the machine availability and the flexibility property of the problem. The developed algorithm is polynomial, as function of the number of operations , flexibilities and availability periods. The proposed approaches is used to develop a lower bound as well as a heuristic for the general problem. As future research, we will apply this method on some benchmarks with practical data and compare this method with a two-phase heuristic developed in a previous work (Zribi and al. , 2005).

## References

- Aggoune, R. (2002). Ordonnancement d'Ateliers sous Contraintes de Disponibilit des Machines. Ph.D. Thesis, Universit de Metz, France.
- Blazewicz, J., J. Breit, P. Formanowicz, W. Kubiak and G.Schmidt. (2001). Heuristic algorithms for the two-machine flowshop problem with limited machine availability. *Omega Journal*, 29, 599-608.
- B. Jurisch (1992). Scheduling Jobs in Shops with Multi Purpose Machines. Ph.D Thesis, Universitt Osnabrck.
- W. Kubiak, J. Blazewicz, P. Formanowicz, J. Breit, and G. Schmidt. Two-machine flow shops with limited machine availability. *European Journal of Operational Research*, 136: 528-540, 2002.
- C. Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9: 395-416, 1996.

- C. Y. Lee. Minimizing the makespan in two-machine flowshop scheduling with availability constraint. *Operations research letters*, 20, 129-139, 1997.
- C. Y. Lee. Two-machine flowshop scheduling with availability constraints. *European Journal of Operational Research*, 114: 420-429, 1999.
- KM. Lee, T. Yamakawa. A Genetic algorithm for general machine scheduling problems. *Int.Conf. on Conventional and knowledge-Based Electronics Systems*, Vol 2 pp60-66 Australia, 1998.
- G. Schmidt. Scheduling on semi identical processors. *Z. Oper.Res.*, A28, 153-162, 1984.
- G. Schmidt. Scheduling independent tasks with deadlines on semi-identical processors. *Journal of Operational research society*, 39, 271-277, 1988.
- Adiri, I., Bruno, J., Frostig, E., and Rinnooy Kan, A. H. G., Single machine flow-time with a single breakdown. *Acta Informatica*, 26, 679-696.
- Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121, 1-15.
- Akers, S. B. and J. Friedman (1955). A non-numerical approach to production scheduling problems. *Operations Research*, 3, 429-442.
- N.Zribi A. El Kamel P. Borne , Minimizing the Makespan for the MPM job-shop with availability constraints . submitted to International Conference on Industrial Engineering and Systems Management IESM 2005, May 16 - 19, Marrakech (Morocco)



# Complexity results for parallel machine scheduling problems with a server in computer systems

S. Guirchoun \*, A. Souhkal \* and P. Martineau \*

*\* Laboratoire d'Informatique - Ecole Polytechnique de  
l'Université de Tours - 64 av J. Portalis, 37200 Tours (France)  
tel: (+33) 247 361 423  
fax: (+33) 247 361 422  
samuel.guirchoun@univ-tours.fr*

## 1 Introduction

We consider deterministic scheduling environment of a computer system composed by a single server and  $m$  identical parallel machines. Each of  $n$  jobs must be loaded by the server and processed without preemption on one of the parallel machines. After a loading or setup, the server is available to perform another loading activity, that is to say a single server can handle only one job at a time and can be considered like a single machine. Moreover, the loading of a job must be immediately followed by its processing. In this environment, we consider several possibilities for the setup time which can require unit or an arbitrary equal time for all jobs. The aim is to find a feasible schedule in this environment with minimum total completion time.

Many results of the last few years are issued from parallel machines scheduling problems with server. [Koulamas, 1996] proposes a beam search heuristic algorithm for an environment with two parallel processors and a single server. [Kravchenko and Werner, 1997], [Hall et al., 2000] and [Brucker et al., 2002] given some complexity results. So a heuristic to minimize the sum of the completion times in the case of unit setup times and arbitrary processing times is proposed by [Kravchenko and Werner, 2001]. In the same way, [Abdekhodae and Wirth, 2002] propose some heuristics to minimize the makespan in the case of equal length jobs. Implicitly, the scheduling problems tackled in these papers consider production environment where, the loading activity is usually considered like a multiprocessor task and requires simultaneously the server and the machine to be performed.

In a computer system, machines dispose of a communication coprocessor which allows them to receive server information at any time. So, the loading activity can be considered like a job that requires only the server to be performed. In this way, the computer system that we consider can be seen as a two-stage hybrid flow shop or multiprocessor flow shop, with a single machine at the first stage, which is the server and  $m$  parallel machines at the second stage, with a no-wait constraint between the two stages. These problems are known to be strongly NP-hard [Gupta, 1988] even for their preemptive version [Hoogeveen et al., 1996]. [Vignier et al., 1999] and [Linn and Zhang, 1999] propose a state-of-the-art survey on hybrid flow shop scheduling problems..

This article is organized as follow : after introduced the notations in Section 2, we show that two-stage hybrid flow shop problems are at least as difficult as the corresponding parallel machine scheduling problems with a single server by a polynomial reduction and we give one theorem and two corollaries for particular cases. Finally, an overview of complexity results for these problems are given for some objective functions.

## 2 Notations

We consider a set  $J$  of  $n$  jobs  $\{i\}_{1 \leq i \leq n}$ . We assume that preemption is not allowed and that each machine can process only one operation at a time. We denote by  $d_i$  the due date and  $r_i$  the release date of job  $i$ .

According to the notation introduced in [Vignier et al., 1999], the hybrid flow shop problem under consideration is denoted by  $FH2, (1, Pm)|nowait|\gamma$ . The first stage contains a single machine and the second stage  $m$  machines. Each job is composed by two operations: the operation  $o_{i,1}$  processed at the first stage and the operation  $o_{i,2}$  processed at the second stage. We denote by  $p_{i,1}$  and  $p_{i,2}$  the processing times of operation  $o_{i,1}$  and operation  $o_{i,2}$ , respectively. The operations at the first stage correspond to the setup of jobs and the second operations at the second stage correspond to the process of jobs. *FH2-problems* refers to a two-stage hybrid flow shop problem.

In the same way, [Kravchenko and Werner, 1997] introduced a notation for identical parallel machine scheduling problems with a single server denoted by  $Pm, S1|s_i|\gamma$ . Each job is composed by two operations: the setup  $s$  processed by the server and a parallel machine and the process  $p$  performed by a parallel machine. We denote by  $s_i$  the setup times and  $p_i$  the processing times of job  $i$ . We note *PS-problems* the parallel machine problems with a single server and *FH2-problems* the two-stage hybrid flow shop problems.

For each job  $i$  of any schedule  $\sigma$  we define for both problems:

$C_i(\sigma)$  refers to the completion time of job  $i$ .

$T_i(\sigma) = \max\{C_i(\sigma) - d_i, 0\}$  the tardiness of job  $i$

$L_i(\sigma) = C_i(\sigma) - d_i$ , the lateness of job  $i$ .

$U_i(\sigma) = 1$  if job  $i$  is completed by its due date, 0 otherwise.

The objective functions contained in the field  $\gamma$  that we consider are :

$C_{max} = \max\{C_i\}$  the latest completion time or makespan of any jobs.

$\sum_{i=1}^n C_i$ , the total completion time of jobs.

$\sum_{i=1}^n w_i C_i$ , the total weighted completion time of jobs.

$\sum_{i=1}^n T_i$ , the total tardiness of jobs.

$\sum_{i=1}^n w_i T_i$ , the total weighted tardiness of jobs.

$L_{max} = \max\{L_i\}$ , the maximum lateness of any jobs.

$\sum_{i=1}^n U_i$ , the number of tardiness jobs.

$\sum_{i=1}^n w_i U_i$ , the number weighted of tardiness jobs.

### 3 Complexity results

In this section, we will present a reduction which leads to complexity results for *FH2-problems*. We have show that two-stage hybrid flow shop problem with a single machine at the first stage,  $m$  identical parallel machines at the second stage, no-wait constraint between two stages and equal processing times at the first stage are at least as difficult as the  $m$  identical parallel machines problem with a single server and equal setup times.

**Theorem 1.** for  $\alpha \in \{m, \emptyset\}$ ,  $\beta \in \{r_i, \emptyset\}$ , setup times  $s_i$  all equal and processing times  $p_i$ , the problem  $P\alpha, S1|s_i = s, \beta|\gamma$  reduces polynomially to  $FH2, (1, P\alpha)|p_{i,1} = s, nowait|\sum$  for all objective functions  $\gamma \in \{C_{max}, L_{max}, \sum C_i, \sum w_i C_i, \sum U_i, \sum w_i U_i, \sum T_i, \sum w_i T_i\}$ .

- 1: Let  $A$  the list where jobs are enumerated in a non decreasing order of their  $r_i$ .
- 2: **while**  $A \neq \emptyset$  **do**
- 3:   Place the first job of  $A$  on the first available machine at the second stage.
- 4:   Remove this job of the list  $A$ .
- 5: **end while**
- 6: Return the solution  $S = \max\{C_{i,2}\}$ .

Table 1: Algorithm for the  $FH2, (1, P\alpha)|p_{i,1} = s, p_{i,2} = p, r_i, \text{nowait}|C_{max}$  problem

By this reduction, some more NP-Hardness results for two-stage hybrid flow shop problems can be derived. For example,  $FH2, (1, P2)|p_{i,1} = p|\sum C_i$  is NP-hard, since the corresponding *PS-problems* is NP-hard [Hall et al., 2000]. In the same way,  $Pm, S1|s_i = 1|\sum C_i$  is solved polynomially, since  $FH2, (1, Pm)|p_{i,1} = 1|\sum C_i$  is solved polynomially [Guirchoun et al., 2004]. More recently, we have proved the complexity of some open problems with the theorem 2 and two corollaries for the makespan criterion.

For the  $FH2, (1, P\alpha)|p_{i,1} = s, p_{i,2} = p, r_i, \text{nowait}|C_{max}$  problem we can see that the schedule where jobs have been placed in the non decreasing order of their release date is optimal.

**Theorem 2.** *The  $FH2, (1, P\alpha)|p_{i,1} = s, p_{i,2} = p, r_i, \text{nowait}|C_{max}$  problem can be solved optimally in a polynomial time by the algorithm 1.*

With the theorem 2 and  $r_i = 0$  for all jobs, we can see that any semi-actif schedule is an optimal solution.

**Corollary 3.** *The  $FH2, (1, P\alpha)|p_{i,1} = s, p_{i,2} = p, \text{nowait}|C_{max}$  problem can be solved optimally in a polynomial time.*

We can also deduce the following corollary.

**Corollary 4.** *The  $FH2, (1, P\alpha)|p_{i,1} = 1, p_{i,2} = 1, \text{nowait}|C_{max}$  problem can be solved optimally in a polynomial time.*

## 4 Synthesis of complexity results

In the end of this paper, Tables 2 presents the complexity results for *FH2-problems*. This table is divided into seven columns. The first column indicates if environment contains 2, 3 or undefined identical parallel machines. The next three columns relate to both processing times and to release date of jobs. The criterion is specified in the fifth column and the complexity of the problem in the sixth column. If the result of the complexity of a problem is deduced from the theorem 2 or 1, or from corollaries 3 or 4, it will be indicated in bold. Otherwise the references of the articles which give the complexity of the problem are specified.

## Conclusion

In this paper, we consider a parallel machine scheduling problem with a server and a hybrid flow shop scheduling problem with a no-wait constraint. We propose a polynomial reduction between these two scheduling problems and we give an overview of complexity results. This polynomial reduction is the beginning of a contribution to show that the two-stage hybrid flow shop problems are at least as difficult as the parallel machine problems with a server for well-known objective functions.

| $m$         | $p_{i,1}$ | $p_{i,2}$    | $r_i$ | Criteria       | Complexity             | References               |
|-------------|-----------|--------------|-------|----------------|------------------------|--------------------------|
| 2           | 1         | any          | no    | $C_{max}$      | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| 2           | $s$       | $\geq s$     | no    | $C_{max}$      | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | 1         | any          | no    | $C_{max}$      | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | 1         | 1            | no    | $C_{max}$      | $\mathbf{O(n \log n)}$ | <b>corollary 3</b>       |
| $\emptyset$ | $s$       | 1            | no    | $C_{max}$      | $\mathbf{O(n \log n)}$ | <b>corollary 4</b>       |
| $\emptyset$ | $s$       | $p$          | yes   | $C_{max}$      | $\mathbf{O(n \log n)}$ | <b>theorem 2</b>         |
| 2           | 1         | any          | no    | $L_{max}$      | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | $s$       | 1            | no    | $L_{max}$      | Open                   | [Hall et al., 2000]      |
| 2           | 1         | any          | no    | $\sum C_i$     | $O(n \log n)$          | [Guirchoun et al., 2005] |
| 2           | $s$       | any          | no    | $\sum C_i$     | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| 3           | 1         | any          | no    | $\sum C_i$     | $O(n \log n)$          | [Guirchoun, 2004]        |
| $\emptyset$ | 1         | any          | no    | $\sum C_i$     | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | 1         | 1            | no    | $\sum C_i$     | $O(n)$                 | [Guirchoun, 2004]        |
| $\emptyset$ | $s$       | 1            | no    | $\sum C_i$     | $O(n)$                 | [Guirchoun, 2004]        |
| $\emptyset$ | 1         | $\geq m - 2$ | no    | $\sum C_i$     | Open                   | [Brucker et al., 2002]   |
| $\emptyset$ | 1         | $\geq m - 1$ | no    | $\sum C_i$     | $O(n \log n)$          | [Guirchoun, 2004]        |
| $\emptyset$ | $s$       | $p$          | yes   | $\sum C_i$     | Open                   | [Brucker et al., 2002]   |
| 2           | 1         | any          | no    | $\sum w_i C_i$ | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | 1         | any          | no    | $\sum w_i C_i$ | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | $s$       | $p$          | yes   | $\sum w_i C_i$ | Open                   | [Brucker et al., 2002]   |
| 2           | 1         | any          | no    | $\sum T_i$     | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | $s$       | 1            | no    | $\sum T_i$     | Open                   | [Hall et al., 2000]      |
| $\emptyset$ | $s$       | $p$          | yes   | $\sum T_i$     | Open                   | [Brucker et al., 2002]   |
| 2           | 1         | any          | no    | $\sum w_i T_i$ | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | $s$       | 1            | no    | $\sum w_i T_i$ | Open                   | [Hall et al., 2000]      |
| $\emptyset$ | $s$       | $p$          | no    | $\sum w_i T_i$ | Open                   | [Brucker et al., 2002]   |
| 2           | 1         | any          | no    | $\sum U_i$     | $\mathcal{NP}$ -Hard   | <b>theorem 1</b>         |
| $\emptyset$ | $s$       | 1            | no    | $\sum U_i$     | Open                   | [Hall et al., 2000]      |
| $\emptyset$ | $s$       | 1            | no    | $\sum w_i U_i$ | Open                   | [Hall et al., 2000]      |
| $\emptyset$ | $s$       | $p$          | no    | $\sum w_i U_i$ | Open                   | [Brucker et al., 2002]   |
| $\emptyset$ | $s$       | $p$          | yes   | $\sum w_i U_i$ | Open                   | [Brucker et al., 2002]   |

Table 2: Synthesis of complexity results for FH2 problems

## References

- [Abdekhodae and Wirth, 2002] Abdekhodae, A. H. and Wirth, A. (2002). Scheduling parallel machines with a single server : some solvable cases and heuristics. *Computers and Operations Research*, 29:295–315.
- [Abdekhodae et al., 2003] Abdekhodae, A. H., Wirth, A., and Gan, H. S. (2003). Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers and Operations Research*.

- [Brucker et al., 2002] Brucker, P., Dhaenens-Flipo, C., Knust, S., Kravchenko, S. A., and Werner, F. (2002). Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5(6):429–457.
- [Guirchoun, 2004] Guirchoun, S. (2004). *Ordonnancement à machines parallèles avec serveur. (in french)*. Thèse de doctorat spécialité informatique, Ecole polytechnique de l’Université de TOURS, Tours (France).
- [Guirchoun et al., 2005] Guirchoun, S., Martineau, P., and Billaut, J.-C. (2005). Total completion time minimization in a computer system with a server and two parallel processors. *Computers and Operations Research*, (32):599–611.
- [Guirchoun et al., 2004] Guirchoun, S., Martineau, P., and Soukhal, A. (2004). Computer system with a server and parallel processors : some solvable cases and heuristics. In *Proceedings of the PMS’2004*, pages 234–237, Nancy.
- [Gupta, 1988] Gupta, J. N. D. (1988). Two-stage, hybrid flowshop scheduling problem. *Operational Research Society*, 39(4):359–364.
- [Hall et al., 2000] Hall, N. G., Potts, C. N., and Sriskandarajah, C. (2000). Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102:223–243.
- [Hoogeveen et al., 1996] Hoogeveen, J. A., Lenstra, J. K., and Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flowshop is np-hard. *European Journal of Operational Research*, 89:172–175.
- [Koulamas, 1996] Koulamas, C. P. (1996). Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers and Operations Research*, 23(10):945–956.
- [Kravchenko and Werner, 1997] Kravchenko, S. A. and Werner, F. (1997). Parallel machine scheduling problems with a single server. *Mathl. Comput. Modelling*, 26:1–11.
- [Kravchenko and Werner, 2001] Kravchenko, S. A. and Werner, F. (2001). A heuristic algorithm for minimizing mean flow time with unit setups. *Information Processing Letters*, 79:291–296.
- [Linn and Zhang, 1999] Linn, R. and Zhang, W. (1999). Hybrid flowshop scheduling : a survey. *Computers and Industrial Engineering*, (37):57–61.
- [Vignier et al., 1999] Vignier, A., Billaut, J.-C., and Proust, C. (1999). Les flowshop hybrides : état de l’art. *RAIRO-RO*, 2(33):117–183.

# **Multi-processor Scheduling**

Full Papers

# A SERVICE LEVEL AGREEMENT PROTOCOL AND TABU SEARCH FOR SCHEDULING ON COMPUTATIONAL GRIDS

<sup>1</sup>D. Ouelhadj, <sup>2</sup>J. M. Garibaldi, <sup>3</sup>R. Sakellariou

<sup>1,2</sup>School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK, {<sup>1</sup>dxs, <sup>2</sup>jmg}@cs.nott.ac.uk

<sup>3</sup>University of Manchester, Oxford Road, Manchester, M13 9PL, UK, <sup>3</sup>rizos@man.ac.uk

**Abstract** The efficient scheduling of independent jobs on distributed Grid resources is one of the most challenging issues in Grid computing. It is a complex problem since resources are geographically distributed having different usage policies and may exhibit highly non-uniform performance characteristics, heterogeneous in nature, and have varying loads and availability.

A recent funded project aims to establish a fundamental new infrastructure for efficient job scheduling on the Grid based on Service Level Agreements (SLAs) and using advanced scheduling techniques. For each job submitted, an SLA is negotiated between the user and the scheduler which contains information on the jobs to execute. In this paper, we propose a negotiation protocol based on the Contract Net Protocol to schedule the SLAs at the global level of the Grid, and the use of tabu search for an efficient scheduling of the SLAs at the local level of the Grid resources. Global scheduling is responsible for resource advertisement, resource discovery, and resource selection. Local scheduling is responsible for determining the order in which the SLAs are executed on the Grid resource.

**Key words:** Grid computing, Grid scheduling, Service Level Agreement, tabu search, batch scheduling heuristics, multi-agent systems, Contract Net Protocol, etc.

## 1. INTRODUCTION

Grid computing has emerged as a new paradigm for next generation distributed computing. A computational Grid is an infrastructure which enables the interconnection of substantial distributed computational resources in order to solve large scale problems in science, engineering, and commerce that cannot be solved on a single system [6]. Scheduling jobs on distributed resources is one of the most challenging issues in Grid computing. It is a complex problem as resources are geographically distributed having different usage policies and may exhibit highly non-uniform performance characteristics, heterogeneous in nature, and have varying loads and availability.

Scheduling jobs on heterogeneous compute resources while trying to minimise some cost function is an NP-complete problem [9]. Most of the Grid scheduling systems [11] such as Nimrod, Nimrod-G, Netsolve, AppLes, Ninf, Condor, Condor-G, LSF, Globus, Legion use simple batch scheduling heuristics, and the common scheduling objective is minimisation of the overall execution time or makespan. Various batch scheduling heuristics have been used [16, 8] including: FCFS, backfilling, list scheduling, smart, Min-min, Max-min, sufferage [2], Xsufferage [4], Qsufferage [15], etc. While many of the Grid scheduling systems use batch-based heuristics for scheduling, these systems do not, however, necessarily aim to find the best scheduling solution. There is little literature on the use of meta-heuristics to generate good quality schedules in Grid computing such as genetic algorithms [1, 3], simulated annealing [1, 17], and tabu search [1]. Jarvis et al. [10] compared FCFS and genetic algorithms to minimise the makespan, and it was found that batch

queuing might be regarded as a worst-case solution. Braun et al. [2] compared the performance of batch queuing heuristics, tabu search, genetic algorithms, and simulated annealing to minimise the makespan. The results showed that genetic algorithms achieved the best results, and the batch queuing heuristics achieved the worst results.

A recent funded project aims to establish a fundamental new infrastructure for efficient job scheduling on the Grid based on Service Level Agreements (SLAs) [12, 13] and using advanced scheduling techniques. For each job submitted, an SLA is negotiated between the user and the scheduler which contains information on the jobs to execute. In this paper, we propose a negotiation protocol to schedule the SLAs at the global level of the Grid, and the use of tabu search for an efficient scheduling of the SLAs at the local level of the Grid resources. Global scheduling is responsible for resource advertisement, resource discovery, and resource selection. Local scheduling is responsible for determining the order in which the SLAs are executed on the Grid resource. Section 2 presents the SLA based Grid scheduling multi-agent infrastructure. Section 3 describes the SLA negotiation protocol for scheduling SLAs at the global level of the Grid, and the scheduling model and the tabu search developed to schedule the SLAs at the local level of the Grid resource. Conclusions are presented in the section 4.

## 2. SLA BASED GRID SCHEDULING MULTI-AGENT INFRASTRUCTURE

The multi-agent infrastructure proposed [13] is organised as a population of cognitive, autonomous, and heterogeneous agents, for integrating a range of scheduling objectives related to Grid compute resources. Figure 1 shows the multi-agent infrastructure for SLA based Grid scheduling. The infrastructure involves three types of agents: User agents, Local Scheduler (LS) Agents, and Super Scheduler (SS) Agents. Three databases are also used in this architecture to store information on SLAs, LSA agents, and resources.

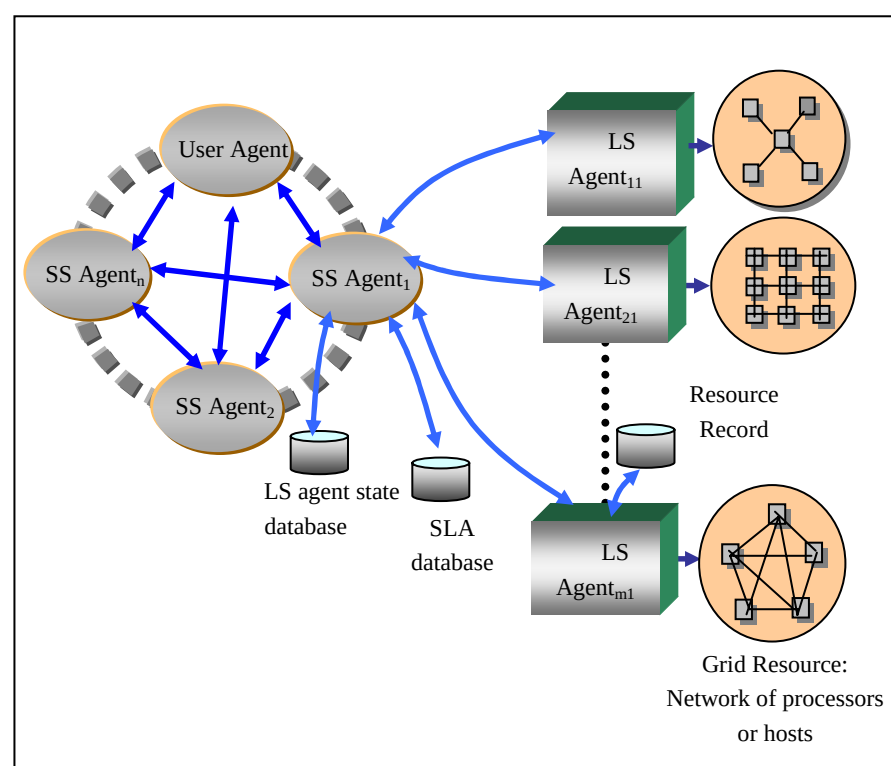


Figure 1. SLA based Grid scheduling system infrastructure



**User Agent:** The user agent requests the execution of SLAs on the Grid and negotiates SLAs with the SS agents. For each job submitted by the user, a SLA is initiated using information on the job supplied by the user and may be refined or additional information may be added in the final agreed SLA at the end of negotiation. The initial SLA submitted by the user describes usually a very high level description of a resource metric containing resource details such as machine information, the range of processors, the expected processing time, etc.

**Local Scheduler Agent:** Each Grid resource within each administrative domain is assigned to an LS agent. The set of Grid resources and their corresponding LS agents constitute a cluster. The LS agents are responsible for scheduling SLAs, usually assigned by the SS agents, at the local level of the Grid resources. However in practice, jobs are unlikely to be arriving from the SS agents, but they can be injected from other means such as PC which is locally connected to the machine that hosts the LS agent.

**Super Scheduler Agent:** SS agents are distributed in the Grid having at most one at every domain. They act as mediators between the user agent and the LS agents. Each cluster of LS agents of the same domain is coordinated by an SS agent. The user agent usually submits SLAs to the SS agents. SS agents negotiate SLAs with the user agent and the LS agents to discover and select a collection of Grid resources that can satisfy SLAs requirements.

**Databases:** The databases used in the architecture are the following:

- *Resource record:* holds information about the compute resources on which the jobs are executed by an LS agent.
- *SLA database:* stores the description of the SLAs.
- *LS agent state database:* provides information about the current availability and capability of Grid resources. This information is used to locate resources, to identify properties of the resources, and for numerous other purposes during the process of translating high-level resource specifications into requests to specific LS agents.

### 3. SLA SCHEDULING

Scheduling SLAs on the Grid involves two stages: global scheduling and local scheduling. Global scheduling discovers and selects the Grid resource to send the SLAs to. Once an SLA is assigned to run at a particular Grid resource, it is then managed by the LS agent at that Grid resource. The local scheduling decides how to schedule the SLAs on the local Grid resource.

#### 3.1 GLOBAL SCHEDULING

Global scheduling is concerned with the discovery and selection of available Grid resources that provide the minimum tardiness and the SLA's compute resource requirements. The global scheduling is performed at the level of the SS agent using a negotiation protocol based on the Contract Net Protocol. The Contract Net Protocol is a high level protocol for achieving efficient cooperation in agent-based systems [5] introduced by Smith [14] based on a market-like protocol. In this protocol, agents exchange task announcements, bids, and awards for distributed task allocation. The SLA negotiation protocol allows the SS agents to negotiate with the user agent and the LS agents the schedule of SLAs on the distributed Grid resources.

The proposed SLA negotiation protocol is a hierarchical bidding mechanism involving two negotiation levels: Meta-SLA negotiation and sub-SLA negotiation.

**1. Meta-SLA negotiation level:** In this level the SS agents and the user agents negotiate meta-SLAs. A meta-SLA contains high-level description of jobs supplied by the user and may be refined or additional information may be added in the final agreed SLA at the end of negotiation. The negotiation steps are the following:

- a<sub>1</sub>. Meta-SLA-request:** The user agent submits a SLA to the nearest SS agent to request the execution of a meta-SLA. The meta-SLA at this stage describes baseline user requirements such as required resources over a time period, expected processing time, job start and end time, etc.
- a<sub>2</sub>. Meta-SLA-announcement:** Upon receiving the meta-SLA-request from the user agent, the SS agent advertises the meta-SLA-announcement to the neighbouring SS agents in the net.
- a<sub>3</sub>. Meta-SLA-bidding:** In response to the meta-SLA-announcement, each SS agent queries the LS agent state database to check the availability of required resources for the job response time to identify suitable set of resources. The SS agents select the best resources which satisfy the earliest execution time and the SLA compute resource requirements. Each SS agent submits a meta-SLA-bid to the user agent. The meta-SLA-bid describes the new estimated response time and the compute resources required.
- a<sub>4</sub>. Meta-SLA-award:** The user agent, upon receiving the meta-SLA-bids, selects the SS agent with the best meta-SLA-bid with the earliest execution time and sends a notification agreement to it. The selected SS agent stores the agreed SLA in the SLA database. The user has now an agreement with the Grid provider to use its resources.

**2. Sub-SLA negotiation level:** in this level the SS agents negotiate sub-SLA with the LS agents. The SS agents decompose the meta-SLA into its low level resource attributes, sub-SLAs which contain low level raw resource description such as processes, memory processors, architecture, memory, disk size, CPU, network, etc. The sub-SLA negotiation steps are the following:

- b<sub>1</sub>. Sub-SLA-announcement:** The SS agent sends the sub-SLA-announcements to the identified suitable LS agents within the cluster under its control.
- b<sub>2</sub>. Sub-SLA-bidding:** The LS agents, upon receiving the sub-SLA-announcements, query the resource record for information on CPU speed, memory size, etc. to identify suitable resources and submit bids to the SS agent.
- b<sub>3</sub>. Sub-SLA-award:** The SS agent evaluates the best sub-SLA in accordance to execution time and resource/requirement match and sends a notification agreement to the LS agent with the best bid. The selected LS agents update the resource record and reserve the resources to execute the job.

### 3.2 LOCAL SCHEDULING USING TABU SEARCH

The local scheduling is performed at the level of the LS agent which schedules the SLAs on the local Grid resource. A Grid resource is considered to be a set of processors or a cluster of workstations.

Consider a Grid resource with  $N$  hosts (processors). At a certain time, there is a set of SLAs denoted by  $S$  that wait to be scheduled to the suitable hosts. For each SLA in  $S$ , we assume an estimation of the execution time is provided by the user.

The scheduling problem is defined by the allocation of a set of independent SLAs,  $S = \{SLA_1, \dots, SLA_s\}$  to a network of hosts  $H = \{H_1, \dots, H_n\}$ .

The expected execution time  $\tau_{ij}$  of  $SLA_i$  on host  $H_j$  is defined as the amount of time taken by  $H_j$  to execute  $SLA_i$ .

The earliest possible start time  $ST_i$  of  $SLA_i$ .

The latest finish time  $FT_i$  of  $SLA_i$ .

The expected completion time  $C_i$  of  $SLA_i$  on host  $H_j$  is defined as the time at which  $H_j$  completes  $SLA_i$ .

The completion time of  $SLA_i$  is defined by  $C_i = ST_i + \tau_{ij}$

The lateness of  $SLA_i$  is defined by  $L_i = \max \{0, C_i - FT_i\}$

The maximum lateness is defined as:

$$L_{max} = \max_{1 \leq i \leq S}(L_i)$$

The objective function is to minimise the maximum lateness:

$$\min L_{max}$$

The Grid scheduling problem is an NP-hard optimisation problem. To solve this combinatorial optimisation-scheduling problem, we propose to use a tabu search method [7] with short-term memory. To generate the initial solution, we propose to use the following batch queuing heuristics: FCFS, Min-min, Max-min, sufferage, and backfilling. Then the solution is improved by using two moves: SLA-swap and SLA-transfer moves. The SLA-swap move swaps two SLAs performed by different processors. SLA-transfer move shifts the SLA to another processor. The search evaluates all the possible moves, and then selects the best move to apply, even if the move yields a worse objective value than the current one until the stopping criterion is reached. Tabu search makes use of a tabu list, where it stores all applied moves in order to avoid cycles, by forbidding a move which reverses a recent one. In practice, each tabu move is initially given a maximum count number, which is the chosen number of forbidden iterations. Then, each time the list is updated the count number of each tabu move is decreased by 1, and on reaching zero, the move is removed from the tabu list, thus becoming valid. When the best move is selected, the tabu list is checked to find out if the move exists. If the best move is not tabu, it is applied to the current solution and then added to the list. If the best move is tabu, then it cannot be applied unless the aspiration criterion holds when the resulting objective value is higher than that of the best solution so far seen. If the criterion fails, then the best move is discarded, the next best move is considered, and the process is repeated.

#### 4. DISCUSSION

In this paper we have described an SLA negotiation protocol for the scheduling of the SLAs at the global level of the Grid. For an efficient schedule of the SLAs to minimise the lateness of SLAs at the local level of the Grid resource, we propose to use tabu search. The proposed novel scheduling methodology based on SLA negotiation protocol and tabu search offer advanced degree of efficiency and flexibility in the negotiation of the SLAs and their scheduling to satisfy user requests. Experiments will be conducted to compare the performance of the schedules generated using simple batch queuing heuristics and the scheduling model using the SLA negotiation protocol and tabu search.

#### ACKNOWLEDGEMENTS

This work is funded by the EPSRC Fundamental Computer Science for e-Science initiative (Grants GR/S67654/01 and GR/S67661/01), whose support we are pleased to acknowledge.

## 5. REFERENCES

1. Abraham, A., Buyya, R., Nath, B. (2000) Nature's heuristics for scheduling jobs on computational Grids. In the 8<sup>th</sup> International Conference on Advanced Computing and Communications, Cochin, India.
2. Braun, T., Siegel, H. J., Beck, N., Hensgen, D., Freund, R. F. (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61, 810-837.
3. Cao, J., Jarvis, S. (2002) ARMS: An agent-based resource management system for Grid computing. *Scientific Programming*. 10, 135-148.
4. Casanova, H., Obertelli, G. (2000) The AppLes parameter sweep template: user level middleware for the Grid. In the Proceedings of the High Performance Networking and Computing of the 2000 ACM/IEEE conference on Supercomputing, Dallas, Texas, United States.
5. Ferber, J. (ed.): *Multi-agent systems: An introduction to Distributed Artificial Intelligence*. Addison-Wesley, London (1999).
6. Foster, I. and Kesselman, C. (eds.) (1998) *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufman Publishers (1998).
7. Glover, F. and Laguna, M. (1997) *Tabu search*. Kluwer Academic Publishers, Boston.
8. Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R. (2000) Evaluation of job scheduling strategies for Grid computing. *Lecture Notes in Computer Science*, pp. 191-202.
9. Ibarra, O. H. and Kim, C. E. (1977) Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of ACM*, 24 (2), 280-289.
10. Jarvis, S. A., Spooner, D. P., Lim Choi Keung, H. N., Nudd, G. R., Cao, J., Saini, S. (2003) Performance Prediction and its use in parallel and distributed computing systems. In the Proceedings of the IEEE/ACM International Workshop on Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems, Nice, France.
11. Krauter, K., Buyya, R., Maheswaran, M. (2002) A taxonomy and survey of Grid resource management systems. 32(2), 135-164.
12. MacLaren, J., Sakellariou, R., Garibaldi, J., Ouelhadj, D. (2004) Towards service level agreement based scheduling on the Grid. In the Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services, in the 14<sup>th</sup> International Conference on Automated Planning & Scheduling, Whistler, Canada, pp. 100-102.
13. Ouelhadj, D., Garibaldi, J., MacLaren, J., Sakellariou, R., Krishnakumar, K. (2005) A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in Grid Computing. Accepted for Publication in the Proceedings of the European Grid Conference, *Lecture Notes in Computer Science*, Springer-Verlag.
14. Smith, R. (1980) The contract net protocol: high level communication and control in distributed problem solver. *IEEE Transactions on Computers*, 29, 1104-1113.
15. Weng, C. and Lu, X. (2003) Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational Grid. *Future Generation Computer Systems*, 21(2), 271-280.
16. Yahyapour, R. (2003) Design and evaluation of job scheduling strategies for Grid Computing, PhD thesis.
17. Yarkhan, A. Dongarra, J. J. (2002) Experiments with scheduling using simulated annealing in a Grid environment. *Lecture Notes in Computer Science, Proceedings of the Third International Workshop on Grid Computing*, pp. 232 – 242.

# HOW GOOD ARE SPT SCHEDULES FOR FAIR OPTIMALITY CRITERIA

Eric Angel,<sup>1</sup> Evripidis Bampis,<sup>1</sup> Fanny Pascual<sup>1</sup>

<sup>1</sup>LaMI, CNRS UMR 8042, Université d'Evry, Tour Evry 2, 523, Place des Terrasses 91000 Evry, France  
{angel, bampis, fpascual}@lami.univ-evry.fr

**Abstract** We consider the following scheduling setting: a set of  $n$  tasks have to be executed on a set of  $m$  identical machines. It is well known that *shortest processing time* (SPT) schedules are optimal for the problem of minimizing the total *sum of completion times* of the tasks. In this paper, we measure the quality of SPT schedules, from an approximation point of view, with respect to the following optimality criteria: *sum of completion times per machine*, *global fairness*, and *individual fairness*.

## 1. Introduction

We are given a set of  $n$  tasks,  $T_1, T_2, \dots, T_n$ , with execution times  $l_1, l_2, \dots, l_n$  and a set of  $m$  identical machines (or processors)  $P_1, P_2, \dots, P_m$ . Given a schedule we denote by  $C_j$  the completion time of task  $T_j$ , and by  $X$  (resp.  $\vec{X}$ ) the completion time vector whose  $i^{\text{th}}$  coordinate is the completion time of task  $T_i$  (resp. where the coordinates of  $X$  have been reordered in non increasing order). For each instance  $\mathcal{I}$  we define  $V(\mathcal{I})$  to be the set of all vectors that are induced by some feasible schedule. One of the most popular optimality criteria in scheduling theory is the *sum of completion times* criterion defined as  $\sum_{i=1}^n C_i$ . An optimal solution to the problem of minimizing the *sum of completion times* can be constructed by using the *shortest execution time* (or Smith's) rule [1]: sort the tasks in non decreasing processing time order and schedule the tasks greedily on the machines following this order: when a machine becomes idle it executes the smallest unscheduled task. A broader class of optimal schedules for the *sum of completion times* criterion is the following class of schedules that are defined using the notion of *rank* [1]. Roughly speaking, a task is at the  $i^{\text{th}}$  rank if its execution time is smaller than or equal to the execution times of the tasks at a largest rank, and largest than or equal to the execution times of the tasks scheduled at a smallest rank.

More formally, we denote by  $\xi$  the ordered collection  $(l_1, \dots, l_n)$  and we assume that the indexing is chosen so that  $l_1 \leq l_2 \leq \dots \leq l_n$ . Our scheduling instance  $\mathcal{I}$  is completely defined by the system  $(\xi, m)$ , where  $m$  is the number of (identical) processors. Let  $\pi_i$  be a collection of processing times defined by  $\pi_k = \{l_n, l_{n-1}, \dots, l_{n-m+1}\}$ ,  $\pi_{k-1} = \{l_{n-m}, \dots, l_{n-2m+1}\}$ ,  $\dots$ ,  $\pi_1 = \{l_{n-(k-1)m}, \dots, l_1\}$ , where  $k = \lceil \frac{n}{m} \rceil$ . The collection  $\pi_i$  is called the  $i^{\text{th}}$  rank of tasks with respect to  $(\xi, m)$ . Consider a schedule obtained by scheduling the tasks rank-by-rank in the order  $\pi_1, \pi_2, \dots, \pi_k$  and such that no two tasks of the same rank are scheduled on the same processor. Notice that any permutation of the tasks of the same rank assures the optimality of the schedule with respect to the *sum of completion times* criterion. This is the family of schedules that we call **SPT schedules** in the sequel.

We are interested to know whether among the **SPT schedules**, which are the optimal schedules for the *sum of completion times* criterion, it is possible to obtain good solutions for the following optimality criteria:

- the *maximum sum of completion times per machine*:  $\max_{1 \leq i \leq m} \sum_{T_j \in P_i} C_j$ .

This measure captures the wish of distributing as much as possible the total sum of completion times among the  $m$  machines of the system.

- the *global fairness* [4]: For two vectors  $X, Y \in V(\mathcal{I})$ , we write  $X \preceq Y$  if  $X_i \leq Y_i$  for all  $i$ . The *global approximation ratio* of  $X$ , denoted by  $c_{gl}(X)$ , is the smallest  $\alpha$  such that  $\vec{X} \preceq \alpha \vec{Y}$  for all  $Y \in V(\mathcal{I})$ . Informally  $c_{gl}(X)$  is the smallest  $\alpha$  for which  $\vec{X}$  is an  $\alpha$ -approximation, in the coordinate-wise sense, to every vector  $Y \in V(\mathcal{I})$ . The best global approximation ratio achievable on the instance  $\mathcal{I}$  is then defined as

$$c_{gl}^*(\mathcal{I}) = \inf_{X \in V(\mathcal{I})} c_{gl}(X).$$

- the *individual fairness*: The *individual happiness factor* compares the completion time of a task with the smallest possible completion time of the same task in any feasible schedule. More formally, we define  $c_{ind}(X)$  to be the smallest  $\alpha$  such that  $X \preceq \alpha Y$  for all  $Y \in V(\mathcal{I})$ . The best approximation ratio achievable on the instance  $\mathcal{I}$  is then defined as

$$c_{ind}^*(\mathcal{I}) = \inf_{X \in V(\mathcal{I})} c_{ind}(X).$$

Our approach is in the same vein as the one of Bruno et al. [3] who considered the following question: *among all optimal schedules for the sum of*

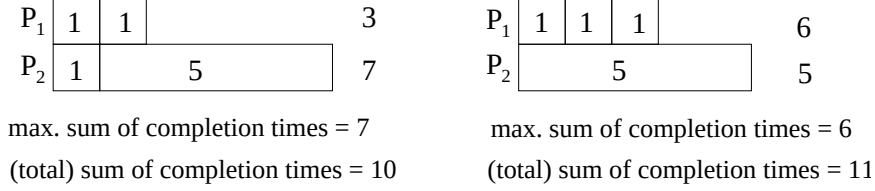


Figure 1. Instance for which the optimal solutions to the problems Minimize  $\sum_{j=1}^n C_j$  and MIN MAX SCT are different.

completion times, is it possible to compute one that minimizes the makespan? (They proved that the problem is  $\mathcal{NP}$ -hard.) For a related problem, see also [6].

**Organization of the paper and our contribution** In Section 2, we consider the problem of *minimizing the maximum completion time per machine* (MIN MAX SCT). We first show that contrarily to the sum of completion times, the problem MIN MAX SCT is  $\mathcal{NP}$ -hard. Furthermore, we show that an SPT schedule is a  $(3 - 3/m + 1/m^2)$ -approximation algorithm for the MIN MAX SCT problem and that there are instances for which any SPT schedule cannot achieve an approximation guarantee better than  $2 - \frac{2}{m^2+m}$ . In Section 3, we consider the *global approximation ratio* and we prove that SPT schedules have an approximation ratio of  $2 - 1/m$  (and that no algorithm can have a better approximation ratio if  $m = 2$ ). Philips et al. [5] presented a 3-approximation for the same problem when release dates are taken into account. For the *individual happiness factor* however the performance ratio cannot be bounded by any constant but we prove that a SPT schedule obtains the best possible performance guarantee. Finally, we focus on a more restricted version of the individual happiness factor where the obtained solutions are compared with respect to the family of SPT schedules and we prove a 2-approximation bound.

## 2. The MIN MAX SCT problem

We first remark that the problem of *minimizing the sum of completion times* and the MIN MAX SCT problem are different. To see that consider the following instance: three tasks of length 1 and a task of length 5 on two identical processors (see Figure 1). The optimal solution for the problem of minimizing the sum of completion times consists in putting two tasks of length 1 on a processor, and the other two tasks on the other processors (total sum of completion times = 10, maximum sum of completion times per processor = 7). The optimal solution of MIN MAX SCT consists in putting the three tasks of length 1 on a processor, and the task of length 5 on the other processor (total sum of completion times = 11, maximum sum of completion time per processor = 6).

## 2.1 Hardness

We prove in this section that the MIN MAX SCT problem is  $\mathcal{NP}$ -hard.

**THEOREM 2.1** MIN MAX SCT is  $\mathcal{NP}$ -hard.

**Proof:** Consider the decision version of the MIN MAX SCT problem:

**MIN MAX SCT problem:** Given a system  $(\xi, m)$ , and a number  $k$ . Does there exist a schedule such that its maximum sum of completion times is smaller than or equal to  $k$ ?

We will show that the PARTITION problem which is known to be  $\mathcal{NP}$ -hard [2] can be phrased in terms of problem MIN MAX SCT.

**PARTITION problem:** Given a collection  $C = \{x_1, x_2, \dots, x_n\}$  of integers. Does there exist a partition  $(A, B)$  of  $C$ , i.e.  $A \cup B = C$  and  $A \cap B = \emptyset$ , such that  $\sum_{x \in A} x = \sum_{x \in B} x$ ?

Given an instance of PARTITION with a collection  $C = \{x_1, x_2, \dots, x_n\}$  of non decreasing integers  $x_1 \leq x_2 \leq \dots \leq x_n$ , we define a system  $(\xi, 2)$  and a limit  $k$  such that the MIN MAX SCT instance admits a solution if and only the instance of PARTITION admits a solution. Let  $k = \frac{3}{2}(\sum_{i=1}^n x_i) - \sum_{i=1}^n \frac{x_i}{n-i+1}$ . We now define the system  $(\xi, 2)$ : we have  $2n$  tasks  $T_1, T_2, \dots, T_{2n}$  to schedule on 2 processors  $P_1$  and  $P_2$ . Let  $l_i$  denote the execution time of task  $T_i$ . Let the execution times  $\xi = \{l_1, l_2, \dots, l_{2n}\}$  be such that  $l_{2j-1} = \sum_{i=1}^{j-1} \frac{x_i}{n-i+1}$  and  $l_{2j} = \sum_{i=1}^j \frac{x_i}{n-i+1}$ , for  $j \in \{1, 2, \dots, n\}$ .

For example, (if  $n \geq 6$ ),  $l_1 = 0$ ,  $l_2 = \frac{x_1}{n}$ ,  
 $l_3 = \frac{x_1}{n}$ ,  $l_4 = \frac{x_1}{n} + \frac{x_2}{n-1}$ ,  
 $l_5 = \frac{x_1}{n} + \frac{x_2}{n-1}$ ,  $l_6 = \frac{x_1}{n} + \frac{x_2}{n-1} + \frac{x_3}{n-2}$ ,  
 $l_{2n-1} = \frac{x_1}{n} + \frac{x_2}{n-1} + \dots + \frac{x_{n-1}}{2}$ ,  
 $l_{2n} = \frac{x_1}{n} + \frac{x_2}{n-1} + \dots + \frac{x_{n-1}}{2} + x_n$

Let us now show that there is a solution of the MIN MAX SCT problem if and only if there is a solution of PARTITION.

If a partition  $(A, B)$  exists for  $C$  then there is a solution of the MIN MAX SCT problem: we can obtain for our system  $(\xi, 2)$  a schedule whose maximum sum of completion times is  $\frac{3}{2}(\sum_{i=1}^n x_i) - \sum_{i=1}^n \frac{x_i}{n-i+1}$ . This schedule can be obtained by assigning tasks  $T_{2j-1}$  and  $T_{2j}$  at rank  $j$ : assign task  $T_{2j}$  to  $P_1$  and  $T_{2j-1}$  to  $P_2$  if  $x_j \in A$ , otherwise assign  $T_{2j}$  to  $P_2$  and  $T_{2j-1}$  to  $P_1$ . Indeed, the execution time of a task added at the  $j^{th}$  rank will be counted  $n - j + 1$  times in the sum of the completion times of its processor. For example, the execution time of the last task of a processor will be counted only



once, whereas the execution time of the first task of a processor will be counted  $n$  times (because there are  $n$  tasks on each processor). Thus the contribution to the sum of completion times per processor of the task  $T_{2j-1}$  (resp.  $T_{2j}$ ), assigned at the  $j^{th}$  rank, is  $(n - j + 1) \times l_{2j-1}$  (resp.  $(n - j + 1) \times l_{2j} = (n - j + 1) \times (l_{2j-1} + \frac{x_j}{n-j+1})$ ). The difference between these two contributions is then  $x_j$ : if  $x_j \in A$  (i.e.  $T_{2j}$  is assigned to  $P_1$ ) then the contribution of the  $j^{th}$  task of  $P_1$  is equal to the contribution of the  $j^{th}$  task of  $P_2$ , plus  $x_j$ . Likewise, if  $x_j \in B$  (i.e.  $T_{2j}$  is assigned to  $P_2$ ) then the contribution of the  $j^{th}$  task of  $P_2$  is equal to the contribution of the  $j^{th}$  task of  $P_1$ , plus  $x_j$ .

Thus, the sum of the completion times of the tasks of  $P_1$  is  $\sum_{j=1}^n T_{2j-1} + \sum_{x_j \in A} x_j$ , and the sum of the completion times of the tasks of  $P_2$  is  $\sum_{j=1}^n T_{2j-1} + \sum_{x_j \in B} x_j$ . Moreover  $\sum_{j=1}^n T_{2j-1} = (n-1)\frac{x_1}{n} + (n-2)\frac{x_2}{n-1} + \dots + \frac{x_{n-1}}{2} = x_1 - \frac{x_1}{n} + x_2 - \frac{x_2}{n-1} + \dots + x_{n-1} - \frac{x_{n-1}}{2} = \sum_{i=1}^n x_i - \sum_{i=1}^n \frac{x_i}{n-i+1}$ . So the sum of the completion times of the tasks of  $P_1$  is  $(\sum_{i=1}^n x_i - \sum_{i=1}^n \frac{x_i}{n-i+1}) + \sum_{x_j \in A} x_j$ , and the sum of the completion times of the tasks of  $P_2$  is  $(\sum_{i=1}^n x_i - \sum_{i=1}^n \frac{x_i}{n-i+1}) + \sum_{x_j \in B} x_j$ .

If there is partition of  $C$  (i.e.  $\sum_{x_j \in A} x_j = \sum_{x_j \in B} x_j = \frac{\sum_{i=1}^n x_i}{2}$ ) then the maximum sum of completion times per processor is equal to  $(\sum_{i=1}^n x_i - \sum_{i=1}^n \frac{x_i}{n-i+1}) + \frac{\sum_{i=1}^n x_i}{2} = \frac{3}{2}(\sum_{i=1}^n x_i) - \sum_{i=1}^n \frac{x_i}{n-i+1} = k$ . Thus, if there is a solution of PARTITION, there is a solution of the MIN MAX SCT problem.

Let us now show that if there is a solution of the MIN MAX SCT problem, then there is also a solution of PARTITION. If the maximum sum of the completion times per processor is smaller than or equal to  $\frac{3}{2}(\sum_{i=1}^n x_i) - \sum_{i=1}^n \frac{x_i}{n-i+1}$ , then we have a SPT schedule. Indeed, the total sum of the completion times of a SPT schedule is  $S = 2 \times (\frac{3}{2}(\sum_{i=1}^n x_i) - \sum_{i=1}^n \frac{x_i}{n-i+1})$ , as we saw it above, and a schedule has a minimum total sum of completion times if and only if it is a SPT schedule [3]. Since the total sum of completion times cannot be greater than twice the maximum sum of completion times per processor, a schedule which is a solution of the MIN MAX SCT problem is a SPT schedule. In any SPT schedule, the tasks of length  $l_{2i}$  and  $l_{2i-1}$  are at the  $i^{th}$  rank, because  $l_1 < l_2 \leq l_3 < \dots < l_{2j-1} < l_{2j} \leq l_{2j+1} < l_{2j+2} \leq \dots < l_n$ . So, in any SPT schedule, the sum of completion times of  $P_1$  minus the one of  $P_2$  is equal to  $\sum_{T_{2j} \in P_1} x_j - \sum_{T_{2j} \in P_2} x_j$ . If there is a solution of the MIN MAX SCT problem with  $k = \frac{3}{2}(\sum_{i=1}^n x_i) - \sum_{i=1}^n \frac{x_i}{n-i+1}$  then the sum of the completion times of  $P_1$  is equal to the sum of the completion times of  $P_2$  (otherwise the sum of completion times of all the tasks would be smaller than  $S$ ), and then  $\sum_{T_{2j} \in P_1} x_j = \sum_{T_{2j} \in P_2} x_j$ : there is a partition of  $C$  and we can construct this partition  $(A, B)$  by placing  $x_j$  in  $A$  if  $T_{2j}$  is assigned to  $P_1$  and  $x_j$  in  $B$  if  $T_{2j}$

is assigned to  $P_2$ . □

## 2.2 Approximation

In this section, we show that an optimal algorithm for the *sum of completion times* criterion gives a 3-approximate solution for the MIN MAX SCT problem. Consider the following algorithm, denoted by  $SPT_{\text{greedy}}$ :

Order tasks by non decreasing execution times. At each step  $i$ , for  $1 \leq i \leq n$ , schedule the current task on the processor which has the smallest completion time.

This algorithm gives an optimal solution for the problem of minimizing the *sum of completion times of the tasks*. Let us show that this algorithm is a  $(3 - \frac{3}{m} + \frac{1}{m^2})$ -approximation algorithm for MIN MAX SCT.

In order to let  $SPT_{\text{greedy}}$  be deterministic when several processors have the smallest execution time, we will refer in the proofs to the following algorithm, which is a greedy SPT algorithm (Proposition 2.1 shows that we add at each step a task on a processor which has the smallest completion time):

Order tasks by non decreasing execution times. At each step  $i$ , for  $1 \leq i \leq n$ , schedule the current task on the processor  $P_{i \bmod m}$ .

PROPOSITION 2.1

- a) *At the beginning of step  $i$  of  $SPT_{\text{greedy}}$ , the processor  $P_{i \bmod m}$  is the processor which has the smallest completion time and the smallest sum of completion times.*
- b) *At the end of step  $i$  of  $SPT_{\text{greedy}}$ , the processor  $P_{i \bmod m}$  is the processor which has the largest sum of completion times.*

**Proof:**

- a) We are at the beginning of step  $i$ . Let  $c$  be the processor  $P_{i \bmod m}$ , and let  $p$  denote its completion time. Let us show that  $c$  has the smallest completion time:
  - For each  $k > i \bmod m$ , processor  $P_k$  has a completion time greater than or equal to  $p$ . Indeed  $P_k$  has the same number of tasks as  $c$  and, for each  $i$  such that  $1 \leq i \leq \text{number of tasks on } c$ , the  $i$ -th task of  $P_k$  is greater than or equal to the  $i$ -th task of  $c$ , by construction.

- For each  $k < i \bmod m$ , processor  $P_k$  has a completion time greater than or equal to  $p$ . Indeed the number of tasks on  $P_k$  is equal to the number of tasks on  $c$  plus one, and, for each  $i$  such that  $1 \leq i \leq$  (number of tasks on  $c$ ), the  $(i + 1)$ -th task of  $P_k$  is greater than or equal to the  $i$ -th task of  $c$ , by construction.

We use the same reasoning to show that  $c$  has the smallest sum of completion times.

- b) We are at the end of step  $i$ . Let  $c$  be the processor  $P_{i \bmod m}$ , and let  $p$  denote its sum of completion times at this step. Let us show that  $c$  has the largest sum of completion times:
- For each  $k > i \bmod m$ , processor  $P_k$  has a sum of completion times smaller than or equal to  $p$ . Indeed  $P_k$  has one task less than  $c$  and, for each  $i$  such that  $1 \leq i < \text{number of tasks on } c$ , the  $i$ -th task of  $P_k$  is smaller than or equal to the  $(i + 1)$ -th task of  $c$ , by construction.
  - For each  $k < i \bmod m$ , processor  $P_k$  has a sum of completion times smaller or equal to  $p$ . Indeed  $P_k$  has the same number of tasks as  $c$  and, for each  $i$  such that  $1 \leq i \leq \text{number of tasks on } c$ , the  $i$ -th task of  $P_k$  is smaller than or equal to the  $i$ -th task of  $c$ , by construction.

□

**PROPOSITION 2.2** *Let  $OPT$  be the maximum sum of completion times of a solution of MIN MAX SCT. We have:*

$$OPT \geq \frac{\text{Min}(\sum_{j=1}^n C_j)}{m},$$

where  $\text{Min}(\sum_{j=1}^n C_j)$  is the optimal value for the problem of minimizing the sum of completion times.

**Proof:** We will prove this proposition by contradiction. Let us suppose that we have  $OPT < \frac{\text{Min}(\sum_{j=1}^n C_j)}{m}$ .

By definition each processor has a sum of completion times smaller than or equal to  $OPT$ :  $\forall i \in \{1, \dots, m\}, \sum_{T_j \in P_i} C_j \leq OPT$ .

So,  $\sum_{j=1}^n C_j = \sum_{i=1}^m \sum_{T_j \in P_i} C_j \leq m \times OPT$ , and  $OPT \geq \frac{\sum_{j=1}^n C_j}{m} \geq \frac{\text{Min}(\sum_{j=1}^n C_j)}{m}$ , a contradiction. □

**THEOREM 2.2** *The algorithm  $SPT_{greedy}$  achieves an approximation guarantee of  $(3 - \frac{3}{m} + \frac{1}{m^2})$  for MIN MAX SCT.*

**Proof:** Before we add the last task  $T_n$ , the processor  $P_x$  on which  $T_n$  will be scheduled has the smallest sum of completion times, denoted by  $p$ , and the smallest completion time, denoted by  $e$  (Proposition 2.1 a). Let  $l_n$  denote the execution time of task  $T_n$ . The processor on which the last task is scheduled has the largest sum of completion times (Proposition 2.1 b), so:

$$\begin{aligned} \max_{1 \leq i \leq m} \sum_{T_j \in P_i} C_j &= \sum_{T_j \in P_x} C_j = p + e + l_n \\ &\leq \frac{\sum_{j=1}^{n-1} C_j}{m} + e + l_n \end{aligned}$$

Since  $SPT_{greedy}$  gives an optimal solution of the minimum sum of completion times problem, and since the completion time of  $T_n$  in  $SPT_{greedy}$  is  $e + l_n$ , we have:

$$\begin{aligned} \max_{1 \leq i \leq m} \sum_{T_j \in P_i} C_j &\leq \frac{\text{Min}(\sum_{j=1}^n C_j) - (e + l_n)}{m} + e + l_n \\ &\leq \frac{\text{Min} \sum_{j=1}^n C_j}{m} + (1 - \frac{1}{m})(e + l_n). \end{aligned}$$

Since  $e + \frac{l_n}{m} \leq OPT$  (because  $e$  is the minimum completion time at step  $n-1$ ), and  $l_n \leq OPT$ , we have:

$$\begin{aligned} e + l_n &= e + \frac{l_n}{m} + \frac{(m-1)l_n}{m} \\ &\leq (1 + \frac{m-1}{m})OPT \\ &\leq (2 - \frac{1}{m})OPT \end{aligned}$$

Moreover, since  $\frac{\text{Min} \sum_{j=1}^n C_j}{m} \leq OPT$  (Proposition 2.2), we have:

$$\begin{aligned} \max_{1 \leq i \leq m} \sum_{T_j \in P_i} C_j &\leq OPT + (1 - \frac{1}{m})(2 - \frac{1}{m})OPT \\ &\leq (3 - \frac{3}{m} + \frac{1}{m^2})OPT. \end{aligned}$$

□

**Lower bound for  $SPT_{greedy}$ .** We now show that  $SPT_{greedy}$  does not achieve an approximation guarantee better than  $2 - \frac{2}{m^2+m}$ . Consider the following instance:  $m$  processors,  $m \times (m-1)$  tasks of length 1 and a task of length  $B = \sum_{i=1}^m i = \frac{m(m+1)}{2}$ .

$SPT_{greedy}$  will schedule  $m-1$  tasks of length 1 on each processor, and the task of length  $B$  will be the last task of the first processor (see Figure 2). The

|                |   |   |   |  |  |
|----------------|---|---|---|--|--|
| P <sub>1</sub> | 1 | 1 | 1 |  |  |
| P <sub>2</sub> | 1 | 1 | 1 |  |  |
| P <sub>3</sub> | 6 |   |   |  |  |

max. sum of completion times = 6

|                |   |   |   |  |  |
|----------------|---|---|---|--|--|
| P <sub>1</sub> | 1 | 1 | 6 |  |  |
| P <sub>2</sub> | 1 | 1 |   |  |  |
| P <sub>3</sub> | 1 | 1 |   |  |  |

max. sum of completion times = 11

Figure 2.  $SPT_{\text{greedy}}$  does not achieve an approximation guarantee better than  $(2 - \frac{2}{m^2+m})$ : example with  $m=3$

maximum sum of completion times is then  $\sum_{i=1}^{m-1} i + (m-1) + B$ , which is equal to  $2(\sum_{i=1}^m i) - 1$ .

An optimal solution for MIN MAX SCT would be the following one: schedule  $m$  tasks of length 1 on each of the  $(m-1)$  first processors, and the task of length  $B$  on the last processor. The maximum sum of completion times in this solution is then  $\sum_{i=1}^m i$ .

So the ratio between the maximum sum of completion times of these two schedules is  $\frac{2(\sum_{i=1}^m i) - 1}{\sum_{i=1}^m i}$ , which is equal to  $2(1 - \frac{1}{m^2+m})$ , which tends towards 2 when  $m$  gets large.

### 3. Fairness measures

In this section we will consider fairness measures in order to compare a schedule given by the greedy SPT algorithm,  $SPT_{\text{greedy}}$ , to any other schedule.

#### 3.1 Global fairness measure

We will first use the fairness measure introduced in [4], namely the *global approximation ratio*. We prove that  $SPT_{\text{greedy}}$  has a global approximation ratio of  $2 - \frac{1}{m}$  for the problem that we consider (and which is denoted by  $P \parallel \text{all}$ ), and that no algorithm can achieve a better global approximation ratio if  $m = 2$ .

**THEOREM 3.1** *One has  $c_{\text{gl}}^*(\mathcal{I}) \leq 2 - \frac{1}{m}$  for all instances  $\mathcal{I}$  of scheduling on  $m$  identical parallel machines ( $P \parallel \text{all}$ ), for any  $m \geq 1$ . Moreover the completion times vector  $X$  of a schedule returned by  $SPT_{\text{greedy}}$  verifies  $c_{\text{gl}}(X) \leq 2 - \frac{1}{m}$ .*

**Proof:** Let us consider an instance  $\mathcal{I}$  of tasks  $T_i$  ( $i \in \{1, \dots, n\}$ ) ordered by increasing lengths. Let MR be the maximal ratio between a schedule  $X_{SPT}$  returned by  $SPT_{\text{greedy}}$  and any schedule  $Y$ :  $MR = \max_Y \frac{\overrightarrow{X_{SPT}}}{\overrightarrow{Y}}$ , where  $\frac{X}{Y}$

means  $\max_i \frac{X_i}{Y_i}$ . So we have  $\forall Y \in V(\mathcal{I}), \frac{\overrightarrow{X_{SPT}}}{\overrightarrow{Y}} \leq \text{MR}$ , and then  $\overrightarrow{X_{SPT}} \preceq \text{MR } \overrightarrow{Y}$ . So  $\text{MR} = c_{\text{gbl}}(X_{SPT})$ . We will show that  $\text{MR}$  cannot be greater than  $2 - \frac{1}{m}$ . Let us consider that this ratio is the  $i^{\text{th}}$  coordinate of  $\overrightarrow{X_{SPT}}$ , divided by the  $i^{\text{th}}$  coordinate of  $\overrightarrow{Y}$  ( $i \in \{1, \dots, n\}$ ).

The worst ratio can be achieved if the completion time of the  $i^{\text{th}}$  task is as large as possible in the  $\text{SPT}_{\text{greedy}}$  schedule. By construction, in a schedule returned by  $\text{SPT}_{\text{greedy}}$ , the  $i^{\text{th}}$  completion time is the completion time of the task  $T_i$ , and  $T_i$  is started after the tasks from  $T_1$  to  $T_{i-1}$ , and before the tasks from  $T_{i+1}$  to  $T_n$ . Since  $\text{SPT}_{\text{greedy}}$  is a greedy algorithm, the worst completion time of the  $i^{\text{th}}$  task is achieved when the  $(i-1)$  first tasks are completed when the  $i^{\text{th}}$  task start to be executed: in this case, the completion time of  $T_i$  is  $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$ . So the worst completion time of  $T_i$  in an  $\text{SPT}_{\text{greedy}}$  schedule is  $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$ .

Let us now find the minimal value which can be taken by the  $i^{\text{th}}$  coordinate of  $\overrightarrow{Y}$ , an ordered completion time vector of a schedule of  $\mathcal{I}$ . Note that this value cannot be smaller than  $l_i$ : indeed  $\overrightarrow{Y}_i$  is the  $i^{\text{th}}$  completion time and is then greater or equal to  $(i-1)$  other completion times, and  $l_i$  is the length of the  $i^{\text{th}}$  smallest task. Note also that  $\overrightarrow{Y}_i$  cannot be smaller than  $\frac{\sum_{j=1}^i l_j}{m}$ : indeed this is the minimum completion time of the  $i$  smallest tasks (when no processor is idle). Therefore, we have:

$$\text{MR} \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{\max\left(\frac{\sum_{j=1}^i l_j}{m}, l_i\right)}.$$

Let  $A$  denote  $\frac{\sum_{j=1}^i l_j}{m}$ . We now consider the two possible cases:

- Suppose that  $l_i \geq A$ , we have:  $\text{MR} \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{l_i} \leq \frac{A - \frac{l_i}{m} + l_i}{l_i} \leq \frac{A}{l_i} + 1 - \frac{1}{m} \leq 2 - \frac{1}{m}$
- Suppose that  $l_i < A$ , we have:  $\text{MR} \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{\frac{\sum_{j=1}^i l_j}{m}} \leq \frac{A - \frac{l_i}{m} + l_i}{A} \leq 1 + \frac{l_i}{A}(1 - \frac{1}{m}) \leq 2 - \frac{1}{m}$

In both cases, we have  $\text{MR} \leq 2 - \frac{1}{m}$ . Since  $\text{MR} = c_{\text{gbl}}(X_{SPT})$ , and since  $X_{SPT}$  is the completion time vector of the  $\text{SPT}_{\text{greedy}}$  schedule of  $\mathcal{I}$ , we proved that a schedule  $X$  returned by  $\text{SPT}_{\text{greedy}}$  verifies  $c_{\text{gbl}}(X) \leq 2 - \frac{1}{m}$ , and so that  $c_{\text{gbl}}^*(\mathcal{I}) \leq 2 - \frac{1}{m}$ .  $\square$

|                |   |   |
|----------------|---|---|
| P <sub>1</sub> | 1 | 1 |
| P <sub>2</sub> | 2 |   |

|                |   |   |
|----------------|---|---|
| P <sub>1</sub> | 1 | 2 |
| P <sub>2</sub> | 1 |   |

completion times vector:  $X_1=(1, 2, 2)$       completion times vector:  $X_2=(1, 1, 3)$

Figure 3. Example where we have  $c_{gbl}^*(\mathcal{I}) = \frac{3}{2} = 2 - \frac{1}{m}$ .

Let us now show that it is not possible to have  $c_{gbl}^*(\mathcal{I}) < 2 - \frac{1}{m}$  for all instances  $\mathcal{I}$  of  $(P \parallel \text{all})$ . Indeed, if  $m = 2$ , it is not possible to have always  $c_{gbl}^*(\mathcal{I}) < \frac{3}{2}$ :

**THEOREM 3.2** *It is not possible to have  $c_{gbl}^*(\mathcal{I}) < \frac{3}{2}$  for all instances  $\mathcal{I}$  of scheduling on two identical parallel machines.*

**Proof:** let us consider a system with two processors,  $P_1$  and  $P_2$ . Let us consider the following instance  $\mathcal{I}$ : a task  $T_1$  of length 1,  $T_2$  of length 1, and  $T_3$  of length 2. Consider the two completion times vector  $X_1 = (1, 2, 2)$  (obtained by putting  $T_1$  and  $T_2$  on  $P_1$  and  $T_3$  on  $P_2$ : see Figure 3) and  $X_2 = (1, 1, 3)$  (obtained by putting  $T_1$  and  $T_3$  on  $P_1$  and  $T_2$  on  $P_2$ ). We have:  $c_{gbl}(X_1) = 2$ , and  $c_{gbl}(X_2) = \frac{3}{2}$ . For each vector  $Y \in V(\mathcal{I})$ , we have  $\vec{X}_1 \preceq \vec{Y}$  or  $\vec{X}_2 \preceq \vec{Y}$ . So for each vector  $Y \in V(\mathcal{I})$ , we have  $c_{gbl}(Y) \geq \frac{3}{2}$ , and so  $c_{gbl}^*(\mathcal{I}) = \frac{3}{2}$ .  $\square$

### 3.2 Individual fairness measure

In this section, we will compare the completion time of each task in a solution given by  $\text{SPT}_{\text{greedy}}$  to the best completion time this task could have in another schedule.

**THEOREM 3.3**  $c_{ind}^*(\mathcal{I}) \leq 1 + \frac{n-1}{m}$  for all instances  $\mathcal{I}$  of scheduling on  $m$  identical parallel machines  $(P \parallel \text{ind\_all})$ , for any  $m \geq 1$ . Moreover the completion times vector  $X$  of a schedule returned by  $\text{SPT}_{\text{greedy}}$  verifies  $c_{ind}(X) \leq 1 + \frac{n-1}{m}$ .

**Proof:** Let us consider an instance  $\mathcal{I}$  of tasks  $T_i$  ( $i \in \{1, \dots, n\}$ ) ordered by non decreasing lengths. It is easy to see that  $c_{ind}(X_{\text{SPT}})$  is the maximal ratio between a schedule  $X_{\text{SPT}}$  returned by  $\text{SPT}_{\text{greedy}}$  and any schedule  $Y$ . We will show that  $c_{ind}(X_{\text{SPT}})$  cannot be greater than  $1 + \frac{n-1}{m}$ . Let us consider that this ratio is the  $i^{\text{th}}$  coordinate of  $X_{\text{SPT}}$ , divided by the  $i^{\text{th}}$  coordinate of  $Y$  ( $i \in \{1, \dots, n\}$ ).

The worst ratio can be reached if the completion time of  $T_i$  is as large as possible in the  $\text{SPT}_{\text{greedy}}$  schedule. Since  $\text{SPT}_{\text{greedy}}$  is a greedy algorithm, this

is achieved when the  $(i - 1)$  first tasks are completed when  $T_i$  start to be executed: in this case, the completion time of  $T_i$  is  $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$ . So the maximal value which can be taken by the  $i^{\text{th}}$  coordinate of  $X_{SPT}$  is  $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$ . The minimal value which can be taken by the  $i^{\text{th}}$  coordinate of  $Y$  is  $l_i$ : this is achieved in a schedule in which  $T_i$  is in the first position. We have:

$$c_{ind}(X_{SPT}) \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{l_i} \leq 1 + \frac{\sum_{j=1}^{i-1} l_j}{m l_i} \leq 1 + \frac{(i-1) l_i}{m l_i} \leq 1 + \frac{n-1}{m}.$$

Since  $c_{ind}(X_{SPT}) \leq 1 + \frac{n-1}{m}$ , we have  $c_{ind}^*(\mathcal{I}) \leq 1 + \frac{n-1}{m}$ .  $\square$

**THEOREM 3.4** *It is not possible to have  $c_{ind}^*(\mathcal{I}) < 1 + \frac{n-1}{m}$  for all instances  $\mathcal{I}$  of scheduling on  $m$  identical parallel machines ( $P \parallel ind\_all$ ).*

**Proof:** Let us consider a system with  $m$  processors and the following instance  $\mathcal{I}$ :  $m + 1$  tasks of length 1. We have  $c_{ind}^*(\mathcal{I}) = 2$  because at least one of the tasks will have a completion time of 2. Since  $c_{ind}^*(\mathcal{I}) = 2 = 1 + \frac{n-1}{m}$ , it is not possible to have  $c_{ind}^*(\mathcal{I}) < 1 + \frac{n-1}{m}$  for all instances  $\mathcal{I}$ .  $\square$

### 3.3 Individual fairness measure among SPT schedules

In this section, we will compare the completion time of each task in an  $SPT_{greedy}$  schedule to the best completion time this task could have in another SPT schedule.

In addition to the notations introduced in the last sections, we define  $V_{SPT}$  as the set of all completion times vectors that are induced by some SPT solutions of the instance. We also define  $c_{SPT}(X)$  to be the smallest  $\alpha$  such that  $X \preceq \alpha Y$  for all  $Y \in V_{SPT}(\mathcal{I})$ . This can be informally viewed as the global approximation ratio of  $X$ : it is the smallest  $\alpha$  for which  $X$  is an  $\alpha$ -approximation to every vector  $Y \in V_{SPT}(\mathcal{I})$ . The best approximation ratio achievable on the instance  $\mathcal{I}$  is then defined as

$$c_{SPT}^*(\mathcal{I}) = \inf_{X \in V_{SPT}(\mathcal{I})} c_{SPT}(X).$$

**THEOREM 3.5**  *$c_{SPT}^*(\mathcal{I}) \leq 2$  for all instances  $\mathcal{I}$  of scheduling on  $m$  identical parallel machines ( $P \parallel ind\_all_{SPT}$ ). Moreover any SPT schedule  $X$  verifies  $c_{SPT}(X) \leq 2$ .*

**Proof:** Let us consider an instance  $\mathcal{I}$  of tasks  $T_i$  ( $i \in \{1, \dots, n\}$ ) ordered by increasing lengths.



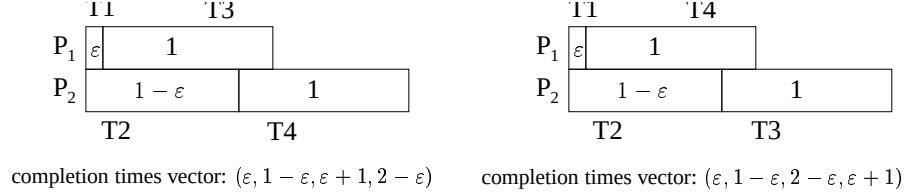


Figure 4. Example where  $c_{SPT}^*(\mathcal{I})$  tends towards 2 when  $\varepsilon$  tends towards 0.

It is easy to see that  $c_{SPT}(X)$  is the maximal ratio between a SPT schedule  $X$  and any SPT schedule  $Y$ . We will show that  $c_{SPT}(X)$  cannot be greater than 2. Let us consider that this ratio is the  $i^{\text{th}}$  coordinate of  $X$ , divided by the  $i^{\text{th}}$  coordinate of  $Y$  ( $i \in \{1, \dots, n\}$ ). Let  $P_{iX}$  (resp.  $P_{iY}$ ) be the processor on which  $T_i$  is scheduled in the solution  $X$  (resp  $Y$ ), and let  $r_i$  denote the rank on which the task  $T_i$  is scheduled.

The worst ratio is reached if the completion time of  $T_i$  is as large as possible in the schedule  $X$ . This is achieved when the tasks before  $T_i$  on  $P_{iX}$  are as large as possible: each largest task on each rank smaller than  $r_i$  is on  $P_{iX}$  in the solution  $X$ . Let  $B_m$  denote the execution time of these tasks.  $B_m$  is the execution time of the tasks on the processor  $P_m$  in an  $\text{SPT}_{\text{greedy}}$  schedule. In the same way, the worst ratio is reached if the completion time of  $T_i$  is as small as possible in the schedule  $Y$ : each smallest task on each rank smaller than  $r_i$  is on  $P_{iY}$  in the solution  $Y$ . Let  $B_1$  denote the execution time of these tasks.  $B_1$  is the execution time of the tasks on the processor  $P_1$  in an  $\text{SPT}_{\text{greedy}}$  schedule. So the difference between the completion time of  $T_i$  in  $X$  and in  $Y$  is equal to  $\delta = B_m - B_1$ . This number is smaller than or equal to the length of each task in the rank  $r_i$ , and then is smaller than or equal to  $l_i$ , the length of  $T_i$  (this property is a direct consequence of Proposition 2.1). We have:

$$c_{SPT}(X) \leq \frac{B_m + l_i}{B_1 + l_i} \leq \frac{B_1 + \delta + l_i}{B_1 + l_i} \leq 2.$$

For any SPT schedule  $X$  of an instance  $\mathcal{I}$ , we have:  $c_{SPT}(X) \leq 2$ , and so  $c_{SPT}^*(\mathcal{I}) \leq 2$ .  $\square$

Let us now show that this bound is the best possible.

**THEOREM 3.6** *Let  $\varepsilon$  be any small number such that  $\varepsilon > 0$ . It is not possible to have  $c_{SPT}^*(\mathcal{I}) < 2 - \varepsilon$  for all instances  $\mathcal{I}$  of scheduling on  $m$  identical parallel machines ( $P \parallel \text{ind\_all}_{SPT}$ ).*

**Proof:** For the ease of presentation we give a proof for a system with two processors. Let us consider the following instance  $\mathcal{I}$ : a task  $T_1$  of length  $\varepsilon$ ,  $T_2$  of

length  $1 - \varepsilon$ , and two tasks  $T_3$  and  $T_4$  of length 1. Assume that  $\varepsilon < 1$ . The completion times vectors corresponding to the two possible SPT schedules are  $X_1 = (\varepsilon, 1 - \varepsilon, 1 + \varepsilon, 2 - \varepsilon)$  and  $X_2 = (\varepsilon, 1 - \varepsilon, 2 - \varepsilon, 1 + \varepsilon)$  (see Figure 4). We have:  $c_{SPT}^*(\mathcal{I}) = c_{SPT}(X_1) = c_{SPT}(X_2) = \frac{2-\varepsilon}{1+\varepsilon} < 2 - \varepsilon$ , which tends towards 2 when  $\varepsilon$  tends towards 0. If  $m > 2$ , the proof is the same except that we add  $m - 2$  tasks of length  $1 - \varepsilon$  and  $m - 2$  tasks of length 1 to the instance described for the case where  $m = 2$ .  $\square$

## References

- [1] R.W. Conway, W.L. Maxwell, and L.W. Miller, *Theory of scheduling*. Addison-Wesley (1967).
- [2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co.
- [3] J. Bruno, E.G. Coffman Jr., and R. Sethi, *Algorithms for minimizing mean flow time*. Proceedings of IFIP Congress 74, 504-510, Stockholm, Sweden, August 5-10, 1974, North-Holland, (1974).
- [4] A. Kumar and J. Kleinberg, *Fairness Measures for Resource Allocation*. Proceedings of 41st IEEE Symposium on Foundations of Computer Science, 75-85 (2000).
- [5] C. Phillips, C. Stein and J. Wein, *Scheduling jobs that arrive over time*. Proceedings of the Fourth Workshop on Algorithms and Data Structures, LNCS 955, Springer, 86-97 (1995).
- [6] C. Stein and J. Wein, *On the existence of schedules that are near-optimal for both makespan and total weighted completion time*. Operations Research Letters 21 (3), 115-122 (1997).

# **A HEURISTIC ALGORITHM FOR SCHEDULING ‘BLOCKED OUT’ UNITS IN CONTINUOUS PROCESSING INDUSTRY : PRELIMINARY RESULTS**

Sumit Kumar Bose and Subir Bhattacharya

*Management Information Systems Group, Indian Institute of Management Calcutta, Joka, Kolkata -700104, India*

**Abstract:** This paper addresses the problem of scheduling cascaded ‘blocked out’ continuous processing units separated by finite capacity storage tanks. The raw materials for the product lines arrive simultaneously on the input side of the first unit. However every unit can process only one product line at a time, thus giving rise to the possibility of spillage of raw material due to limited storage capacity. The need to process multiple product lines, and the added constraint of multiple intermediate upliftment dates aggravate the problem. This problem is quite common in petrochemical industry. The paper provides a MINLP formulation of the problem. However, for any realistic scheduling horizon, the size of the problem is too large to be solved by standard packages. We have proposed a heuristic propelled depth first branch and bound algorithm to help the planners in tackling the problem. The suggested algorithm could output near optimal solutions for scheduling horizons of 30 time periods when applied on real life situations involving 3 units and 3 product lines. The preliminary results reported here are quite encouraging.

**Key words:** Process scheduling, Heuristic search, Depth first branch and bound, Continuous processing

## **1. INTRODUCTION**

The scheduling problem addressed in this paper occurs quite frequently in refineries and other chemical processing industries. In a refinery set up, for example, streams get split into intermediate products each of which then possibly moves through a network of processing units and blending points before being converted to finished products. As a result, the scheduling of a unit depends on the scheduling of upstream units and influences the scheduling of downstream units. Some of the units in the network may be responsible for producing a range of products. In this situation, the unit usually works in a ‘blocked-out’ fashion, i.e., at any point of time it processes only one product line or stream. However, arrival of raw materials for other products is simultaneous, being outputted from some upstream unit(s). Inputs for product lines, other than the one currently being processed, must either be stored in tanks for future processing, or it will get ‘spilled’ or wasted. The spilled amount usually gets downgraded to substantially lower valued products. Spillage can be reduced by quick changeovers. But a changeover from one product line to another has its associated cost and time. Thus, given fixed capacity storage tanks, there is always a trade-off between spillage and changeover while processing multiple products on the same processing unit. Sometimes, these blocked-out units are cascaded one after the other – separated by fixed capacity storage tanks for each product line – and thus complicates the scheduling decision further. The problem of scheduling gets aggravated by the requirement of meeting the upliftment schedule with its associated penalty for not producing the required amounts of finished products within the due dates. The objective of this paper is to study similar scheduling problems and to suggest heuristic propelled search scheme to get ‘near optimal’ solutions.

The processing units under consideration are continuous processing units. In a batch processing unit like a reactor, inputs in right amount and proportion are fed into the unit, ‘treated’ for a fixed amount of

time in a non-preemptive style, and outputs are obtained after the complete ‘batch’ gets processed. Scheduling of batch processing units addresses the issue of determining the optimal sequence in which the product batches should be produced. On the other hand, in a continuous processing unit, like a distillation column for example, inputs for a product line are fed in continuously at a fixed rate and the outputs flow out simultaneously at a fixed rate. Here, one product line can be processed in a number of stretches during the scheduling horizon interleaved with the processing of other streams. The length of a stretch is not fixed and each such stretch is called *run length* (see figure 1).

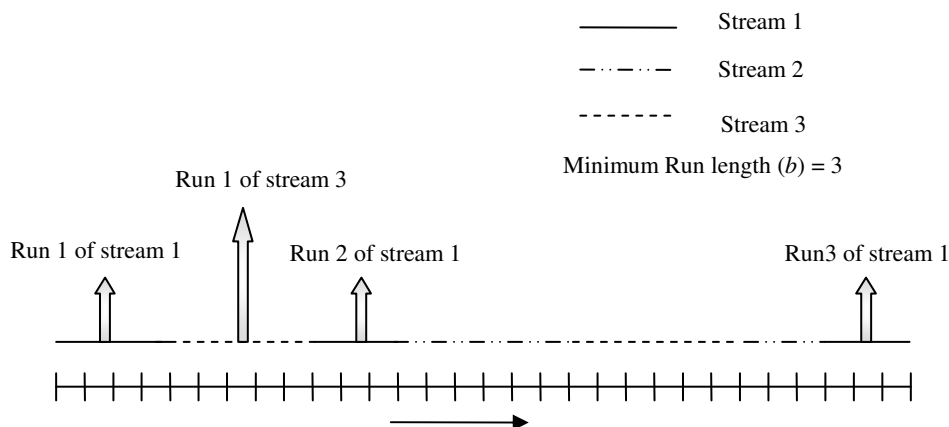


Figure 1. Example of Run Lengths in a unit

Longer stretches of run are desirable to reduce changeover cost, while shorter stretches may reduce spillage cost and upliftment failure penalty. The scheduling problem aims to find out an optimal mix of product sequences of varying run lengths so as to minimize the total cost/penalty. The authors encountered a similar problem while scheduling production of different categories of lube oils using a sequence of blocked-out units in a refinery.

Section 2 of the paper gives a broad review of the literature in this area. The problem scenario under consideration is explained in detail in section 3 and the corresponding mathematical model is discussed in section 4. Section 5 elaborates the heuristic solution procedure suggested for solving the problem. Our computational experience with the algorithm is summarized in section 6. Concluding remarks are presented in section 7.

## 2. LITERATURE REVIEW

The problem of scheduling in the context of process industry, as opposed to discrete manufacturing industry, has received considerable attention of the researchers only in the recent past. However, the bulk of the research work addresses the problem of scheduling batch processing systems. Work on scheduling of continuous processing systems is picking up only very recently. Rippin (1993) and Sargent (2004) describe the general status of process systems. Reklaitis (2000) outlines the generic families of existing methodologies for process scheduling operations. Shah (1998) extensively reviews the techniques available for optimizing the production schedules in the process industries. Grossmann et al. (1996)

discusses the mixed integer mathematical programming methods for the scheduling of batch processes. Kondili et al. (1993), in their seminal paper propose 'state task network' representation for batch processes and formulate a mixed integer linear program based on discrete time representation for handling different kinds of scheduling problems arising in batch processing plants. The computational issues related to the MILP formulation presented by Kondili et al. are discussed in Shah et al. (1993). Pinto and Grossmann (1998) summarizes the various assignment and sequencing models used in the process industry. Frauendorfer and Konigsperger (1996) discusses the various scheduling objectives and performance evaluation criteria in the context of the chemical process industry. Graham et al. (1979) and Garey and Johnson (1979) have shown that almost all the scheduling problems are NP-complete, implying that a large number of feasible solutions have to be examined before chancing upon a provably optimal solution. Thus efficient solution procedures can only result from applying problem specific heuristics. Kudva et al. (1994), Ku and Karimi (1990, 1991), Wiede and Reklaitis (1987a, 1987b), Blomer and Gunther (1998) have suggested application of heuristic techniques to various multi product batch scheduling problems. Schilling and Pantelides (1996), Zhang and Sargent (1996), Mendez and Cerda (2002) have developed continuous time formulations for the short term scheduling problem in batch processes. Ierapetritou and Floudas (1998) and Ierapetritou et al. (1999) formulate mathematical models for continuous processes and account for multiple intermediate due dates. Jain and Grossmann (1998) builds a mathematical model for cyclic scheduling of continuous processing plants with parallel units and decaying performance. Alle and Pinto (2004) develop a mathematical model for cyclic scheduling of multi product multistage continuous processing plants. Jia and Ierapetritou (2004) consider the short term scheduling of refinery operations and develop mixed integer models based on continuous time formulations. To the best of our knowledge there is no effective solution procedure in the scheduling literature to deal with the scheduling problem in multi stage, multi product continuous facilities with finite intermediate storage, multiple upliftment due dates and simultaneous arrival of inputs. This work is an attempt to advance the research in this direction.

### 3. PROBLEM SCENARIO

The set of units under consideration consists of a sequence of  $n$  'blocked-out' processing units,  $U_1, U_2 \dots U_n$ , buffered by fixed capacity storage tanks to hold intermediate products (Figure 2). Each unit can process only one product line (also referred to as *stream*) at a time. The presence of intermediate storage tanks obviates the need for the units to process the same product in tandem. The finished products coming out of the last unit ( $U_n$ ) also get stored in fixed capacity tanks to be uplifted according to some pre-specified upliftment schedule. The inputs for the product lines arrive at the input tanks of Unit  $U_1$  simultaneously at constant rates. The rates at which these inputs arrive depend on factors that are beyond the control of this block of units. Each of these streams gets stored in a fixed capacity tank if there is room (*ullage*) in it; otherwise it spills, and is downgraded to lower valued products. Each product line has its exclusive set of tanks before and after every processing unit. Thus, each unit processes one product at a time taking its input from the corresponding input tank if it has enough stock and depositing the output into corresponding output tank if it has enough ullage. Note that spillage is not allowed for intermediate or finished products. In essence, spillage can occur only at the input of  $U_1$  because the inputs for all the product lines are arriving simultaneously and also because these units have no control on the rates at which these inputs arrive.

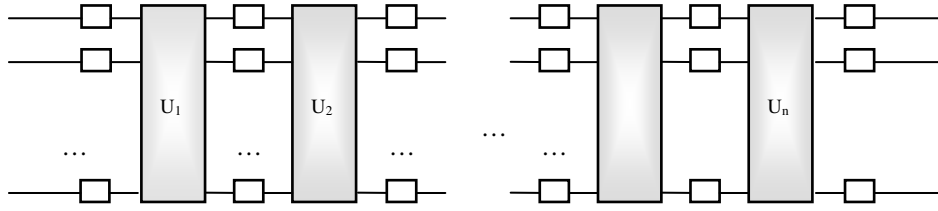


Figure 2. Continuous processing plant with  $n$  stages and  $m$  products

We consider  $m$  product lines,  $P_1, P_2 \dots P_m$ , each of which requires to be processed by the  $n$  units in succession. The processing capacity (known as the feedrate measured in MtPD, Metric tons per day) of a unit for a product line is considered fixed, but varies from product to product. The processing of a unit involves splitting the feed with the help of reagents into intermediate streams according to a yield percentage fixed for a product line for the unit. The intermediate streams that are relevant for the final products under consideration get deposited in the tanks on the output side. What happens to the other streams is beyond the purview of this discussion. For our purpose, if  $p_{ij}$  is the yield percentage corresponding to product  $P_i$  in unit  $U_j$ , then  $x$  unit of input to  $U_j$  gives rise to  $x * p_{ij}$  units of output from  $U_j$ .

Given the above set-up, the problem is to schedule the production of the  $m$  products in each of the  $n$  units so as to minimize the total penalty. Three factors contribute to the penalty – the per unit spillage penalty for the raw material of each product line, the cost for changeover, and the per unit penalty for failing to meet the upliftment schedule.

#### 4. THE MATHEMATICAL MODEL

The objective of the proposed model is as follows: given a scheduling horizon of  $H$  time periods, decide the product lines to be processed in each time period in each unit so as to minimize the total cost. A product  $i$  taken up for production in unit  $j$  must run for at least  $b$  consecutive time periods. This restriction is imposed to efficiently utilize the minimum amount of reagents that need to be added when changing over to product  $i$ . The number of consecutive time periods during which a product is processed (*run*) in a unit is called the *run length*. Thus, run length of a product in a unit must be greater than or equal to the minimum run length  $b$ . For simplicity, the products currently running at the units are assumed to have completed their minimum run lengths at the beginning of the scheduling horizon. That is, each unit is free to changeover to some other product if it so desires.

Changing over from one product to another requires some set up time. However, for simplicity, we assume the changeover to be immediate. The changeover costs are assumed to be sequence independent. Product-wise feedrates and yield percentages for the units are fixed data. The arrival rates of the inputs in front of (on the left of)  $U_1$  and the upliftment schedule for the planning period are known at the time of deciding the units' schedule. The same product can be uplifted several times by different quantities during the given  $H$  time periods. The upliftment schedule may specify upliftment of more than one product at the same time period. All the units are assumed to be up and running for the entire scheduling horizon.

To facilitate our discussion, the storage tanks are assumed to belong to certain *levels*. The tanks in front of  $U_1$  belong to 'level 0' and the tanks immediately after a unit  $U_k$  are said to be in 'level  $k$ '.

Table 1. Notations

**Sets:**

$i$  : Stream identifier,  $i \in [1, 2, \dots, m]$

$j$  : Unit identifier,  $j \in [1, 2, \dots, n]$

$t$  : Time period,  $t \in [1, 2, \dots, H]$

$l$  : Level of tanks,  $l \in [0, 1, 2, \dots, n]$

**Parameters:**

$r_i$  : Per unit spillage penalty of stream  $i$ .

$a_i$  : Rate of arrival of raw material of product line  $i$ .

$C_{il}$  : Capacity of tanks corresponding to product line  $i$  and level  $l$ .

$q_j$  : Cost of changeover from one stream to another in unit  $j$ .

$u_i$  : Per unit penalty for unfulfilled demand of stream  $i$ .

$p_{ij}$  : Percentage yield of stream  $i$  in unit  $j$ .

$dem_{it}$  : Quantity of stream  $i$  to be uplifted at the beginning of period  $t$ .

**Variables:**

$S_{it}$  : Quantity of material in tank for product  $i$  at level  $l$  at the start of time period  $t$ .

$Y_{ijt}$  = 1 if stream  $i$  is processed on unit  $j$  in time period  $t$ ; 0 otherwise.

$X_{it}$  = 1 if  $S_{i0t} + a_i > C_{i0}$  and  $Y_{ijt} = 0$ ; 0 otherwise.

$V_{ijt}$  = 1 if  $Y_{ijt} = 1$  and  $Y_{ijt+1} = 0$ ; 0 otherwise.

$W_{it}$  = 1 if  $(dem_{it} - S_{int}) > 0$ ; 0 otherwise.

We will use the notations of Table 1 to formulate the problem. The short term scheduling problem of continuous processing blocked out units can be formulated as an MINLP as follows:

Minimize

$$Z = \text{Min} \sum_t \sum_i r_i (S_{i0t} + a_i - C_{i0}) X_{it} + \sum_t \sum_j q_j (\sum_i V_{ijt}) + \sum_t \sum_i u_i (dem_{it} - S_{int}) W_{it} \quad (1)$$

Subject to:

$$\sum_{t'=t}^{t-b+1} Y_{ijt'} \geq b V_{ijt} \quad \forall t, j, i \quad (2)$$

$$S_{i0t+1} = X_{it} C_{i0} + (1 - X_{it}) (S_{i0t} + a_i) - f_{i1} Y_{i1t} \quad \forall i, t \quad (3)$$

$$S_{ijr+1} = S_{ijt} + f_{ij} p_{ij} Y_{ijt} - f_{ij+1} Y_{ij+1t} \quad \forall i, t, j = 1, \dots, n-1 \quad (4)$$

$$S_{int+1} = (1 - W_{it}) (S_{int} - dem_{it}) + f_{in} p_{in} Y_{int} \quad \forall i, t \quad (5)$$

$$0 \leq S_{ilt} \leq C_{il} \quad \forall i, l = 0, \dots, n \quad (6)$$

$$\sum_i V_{ijt} \leq 1 \quad \forall j, t \quad (7)$$

$$\sum_i Y_{ijt} \leq 1 \quad \forall j, t \quad (8)$$

$Y_{ijt}$ s are the binary decision variables that define the scheduling of streams in the units.  $Y_{ijt}$  takes the value 1 if stream  $i$  is processed in unit  $j$  during time period  $t$ . The binary variables  $X_{it}$ ,  $V_{ijt}$  and  $W_{it}$  represent spillage, changeover and upliftment failure respectively during time period  $t$ . If stream  $i$  spills during time period  $t$ , then  $X_{it}$  is 1, otherwise 0. The value of  $V_{ijt}$  is 1 when stream  $i$  is processed in unit  $j$  during time period  $t$  but is not processed during time period  $t+1$ , indicating changeover to some other stream  $i'$  in period  $t+1$ .  $W_{it}$  equals 1 when the amount of finished product  $i$  available at time  $t$  for upliftment falls short of the quantity demanded at that point of time.

The objective is to develop a schedule that would lead to the least overall cost. The terms in the objective function express the cost of spillage, the cost of changeover and the upliftment failure cost respectively. Equation (2) ensures that changeover cannot occur in a unit  $j$  till the *minimum run length* of the stream being processed in the unit is over. Equation (3) updates the stock positions in tanks dedicated for storing the raw materials to be used as feed to the unit,  $U_1$ . The stock in the tank corresponding to stream  $i$  at the start of the time period  $t+1$  is equal to the stock at the start of time period  $t$  adjusted by the amount of raw material arriving in period  $t$ , the spillage, if any, in period  $t$  and the withdrawal, if the stream is processed by unit  $U_1$  in period  $t$ . Equation (4) updates the stock positions of the intermediate tanks at the beginning of each period. Equation (5) models the update of the stock positions of the finished product tanks. Equation (6) constrains the stock of a tank not to exceed the capacity of the tank at any point of time. Equation (7) ensures that a unit cannot change to more than one stream at any time period. Equation (8) ensures that at most one stream is assigned to a unit for processing during period  $t$ .

Employing linearization techniques the above mathematical model was reformulated as an MILP and solved using GAMS/XA on a 2.4 GHz Pentium 4 workstation with 1 GB RAM. Obtaining feasible schedules for horizons of more than 8 days (time periods), with  $m=3$  and  $n=3$ , proved difficult even with very high values for resource count and iteration limit. Since the planners are frequently interested in deciding the schedule of the units for a longer period of time, say, 30 days, the above model, as of now, does not serve its purpose. To meet their requirement, we need to look for some other mechanism, which can provide 'reasonably good' solutions though not provably optimal. To that end we propose a heuristic algorithm below such that near optimal solutions can be obtained for scheduling horizons up to 30 days in reasonable time.

## 5. HEURISTIC SOLUTION PROCEDURE

As mentioned, the above mathematical formulation becomes computationally expensive for scheduling horizons of any practical significance. To take care of this problem, we propose below a heuristic depth-first branch-and-bound (DFBB) solution procedure for a simplified version of the problem. The scheme has been tried out using real-life data of a situation where there are 3 product lines ( $m=3$ ) to be processed using 3 cascaded blocked-out units ( $n=3$ ) for a planning horizon of 30 days ( $H=30$ ). Our computational experience in section 6 will demonstrate that this heuristic scheme can give optimal/near-optimal solutions in reasonable time for real life situations.

The search space is a tree where a node represents a feasible partial schedule till the time period  $t$ . A feasible partial schedule till a period  $t$  contains feasible assignments of product lines (*streams*) to the processing units for every time period up to and including  $t$ . An assignment of a stream to a unit at any time period is said to be *feasible* if the stream has enough input stock and output ullage for processing the stream in this unit for one time period. A partial schedule till period  $t$  can be extended to partial schedules till period  $t+1$  by feasibly assigning streams to the units for the  $(t+1)^{\text{th}}$  time period, given the assignments till period  $t$ . Thus, given 3 streams and 3 units, a node in the search tree can give rise to a maximum of 27 child nodes, all of which, however, may not be feasible.



The root node of the search tree represents partial schedule till period 0. A leaf node is one which contains partial schedule till the scheduling horizon  $H$  and hence represents a feasible solution to the scheduling problem. The tree is unveiled and traversed in a depth-first manner in its quest for the solution node with the least cost.

The above search tree can have a maximum of  $27^{30}$  leaf nodes for a 30 period scheduling horizon. Obviously, an exhaustive search scheme is not computationally tenable. We augment the depth-first search using the following mechanisms:

- a) An admissible heuristic for the upliftment failure penalty;
- b) An admissible heuristic for the spillage cost;
- c) An efficient node ordering; and
- d) A 'reasonably good' initial solution that provides a tight upper bound.

In the sub-sections below we elaborate on each of the above schemes.

The heuristic scheme discussed in this paper does not take into account the changeover cost. That is, the cost function being minimized is the sum of spillage costs and the upliftment failure penalties only. As explained below, our heuristics for spillage cost and upliftment failure penalty at a node give us good lower bound on the total cost of the potential extensions of a partial schedule and, hence, efficiently prune provably sub-optimal branches of the search tree. However, given a partial schedule, we have so far not been able to come up with a good admissible heuristic to get a lower bound on the number of changeovers for the extensions of the schedule. Work is currently going on to factor in changeover cost as well. This paper discusses the efficacy of the rest of the scheme and preliminary results obtained thereon.

## 5.1 Heuristic for Upliftment failure penalty

Upliftments for the various finished products are met from the output tanks of the last unit,  $U_3$ . Upliftment time periods and quantities are known a priori for the entire scheduling horizon. Please recall that  $dem_{it}$  is the quantity of the  $i^{th}$  stream to be uplifted at the beginning of time period  $t$ . Material in stock,  $S_{i3t}$ , in the output tank of the  $i^{th}$  stream at the end of time period  $t-1$  is considered available for meeting the upliftment at the  $t^{th}$  time period. If the stock contains more than  $dem_{it}$ , only  $dem_{it}$  amount is uplifted and if it is less,  $S_{i3t}$  is the amount uplifted and the penalty incurred is  $(dem_{it} - S_{i3t}) * u_i$  where  $u_i$  is the penalty per unit for the shortfall for the  $i^{th}$  product. Shortfall in one upliftment cannot be compensated for by providing more during subsequent upliftments.

Let us assume that we have a partial schedule till time period  $t-1$ . To keep our heuristic admissible, we assume that there is no dearth of stock for any of the streams on the input side of unit  $U_3$ . That is,  $U_3$  can process any stream by any amount so long as there is ullage at the output tanks. It is easy to see that, during actual processing, penalty can only go up due to non-availability of the corresponding input stocks. The calculation of the heuristic consists of building possible scenarios of meeting the upliftments by assigning periods of  $U_3$  processing to various streams, calculating the cost of each of the scenarios, and picking up the least cost one. It is worth bearing in mind that assigning time periods to a stream has nothing to do with the scheduling of the stream which includes sequencing as well. It just sets aside a number of time periods for processing of the stream.

To explain the process of calculating the heuristic for upliftment failure penalty we will use the notations given in Table 2 for each finished product (stream)  $i$ .

Table 2. Notations for Heuristic Calculations

---

|               |                                                                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $dt_{ik}$ :   | Upliftment time period for stream $i$ for the $k^{\text{th}}$ remaining upliftment                                                                                                                  |
| $n_i$ :       | Number of remaining upliftments of stream $i$ on time periods $dt_{ik} (dt_{ik} > t)$ , for $k = 0, 1, 2, \dots, n_i$                                                                               |
| $h_{ik}$ :    | Time periods available for meeting the $k^{\text{th}}$ upliftment of stream $i$ , that is number of unassigned time till $dt_{ik}$                                                                  |
| $h$ :         | Time periods available till the last upliftment in the schedule.                                                                                                                                    |
| $d_{ik}$ :    | The number of $U_3$ processing time periods required to meet the $k^{\text{th}}$ upliftment given the tank stock after $(k-1)^{\text{th}}$ upliftment. $d_{ik}$ need not necessarily be an integer. |
| $\hat{d}_i$ : | Total amount of $U_3$ processing time periods required to meet all the remaining upliftments. $\hat{d}_i$ need not necessarily be an integer.                                                       |

---

We start by assigning periods for the ‘most vulnerable’ stream. The ‘most vulnerable’ stream is the one for which the product  $f_{i3} * p_{i3} * u_i$  is maximal among all the streams. Conceptually, if all the finished products require one more period of processing and only one period of processing is available, then the processing of this stream will minimize the total upliftment penalty. Streams are taken up for assignment in decreasing order of vulnerability. For a stream currently under consideration, we keep on assigning processing periods for meeting the quantities needed for *all* its upliftments starting from the next upliftment. When all the upliftments of a stream are taken care of, we repeat the same process for the next vulnerable stream and so on.

We now elaborate the scheme for developing the possible scenarios for assigning time periods. Let us assume, for the time being, that the stream running in  $U_3$  at  $t-1$  has completed its minimum run length  $b$ , that is,  $U_3$  is free to changeover to some other stream if it so desires. Let us assume that currently we are considering the 1<sup>st</sup> upliftment of the  $i^{\text{th}}$  stream after time period  $t$ , and  $i^{\text{th}}$  stream was not being processed in  $U_3$  at  $t-1$ . Since there *has* to be a changeover to stream  $i$  to meet its 1<sup>st</sup> upliftment (assuming  $d_{i1} > 0$ ), we can exploit the fact that any assignment of periods to stream  $i$  must be at least for the minimum run length  $b$ .  $d_{i1}$ , the number of processing days required, given the output tank stock at the beginning of period  $t$ , need not necessarily be (and is usually not) an integer. The number of periods to be assigned will depend on many factors, combinations of which give rise to the various scenarios. If  $i^{\text{th}}$  stream is not the one that was being processed in time period  $t-1$ , then the leaf nodes of figure 3 give all the possible assignments that need to be considered. For example, if  $d_{i1}$  is less than or equal to the number of available periods for processing the  $i^{\text{th}}$  stream for the 1<sup>st</sup> upliftment ( $h_{i1}$ ), we need to first check whether the demand is enough to be assigned for minimum run length  $b$ . If yes, we have to try two scenarios – one by assigning  $\text{ceil}(d_{i1})$  periods and the other with  $\text{floor}(d_{i1})$ . However, if the needed processing periods is less than the minimum run length  $b$ , we need to consider  $\hat{d}_i$ , the total number of processing periods required for meeting all the remaining upliftments. If the total requirement is also less than or equal to  $b$ , we need to try two scenarios by assigning 0 and  $b$  time periods to stream  $i$ . Otherwise, depending on whether  $\hat{d}_i$  is greater than two times  $b$  or not we will either try two scenarios by assigning 0 or  $b$  days, or we will construct four scenarios assigning 0,  $b$ ,  $\text{ceil}(\hat{d}_i)$  and  $\text{floor}(\hat{d}_i)$  periods to stream  $i$ .

For apportioning processing periods for subsequent upliftments of the same stream  $i$ , however, we cannot impose the restriction of minimum run length since it will depend on actual sequencing of the streams. For subsequent upliftments, we just compare  $d_{ik} (k > 1)$  with corresponding  $h_{ik}$ . If  $d_{ik}$  is greater than  $h_{ik}$ , we assign  $h_{ik}$  time periods to stream  $i$ , else we consider the scenarios of assigning  $\text{ceil}(d_{ik})$  and  $\text{floor}(d_{ik})$ . We have to follow the same method to generate scenarios for meeting upliftments of the stream that was being processed on period  $t-1$ .

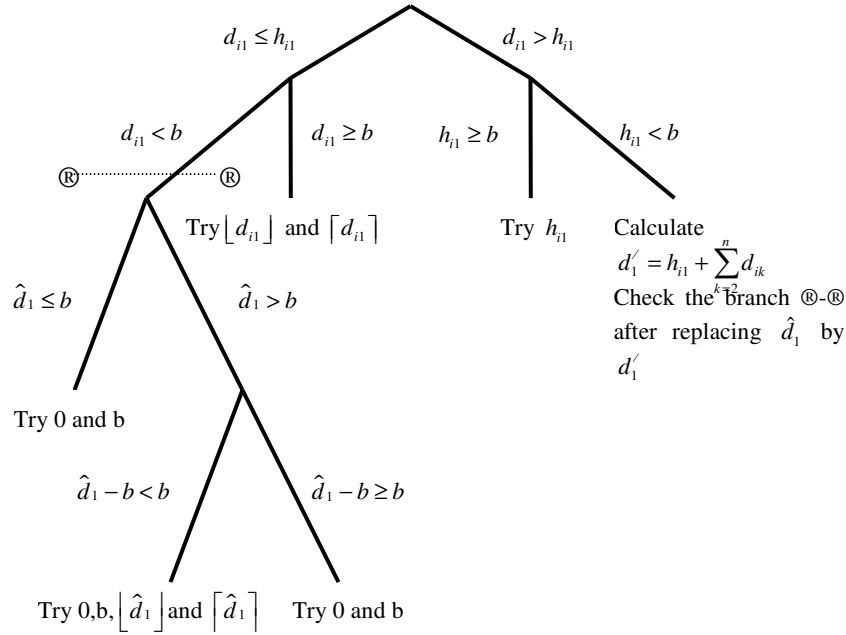


Figure 3. Decision tree for heuristic upliftment

After each assignment of time periods to an upliftment, we need to carefully update the available processing periods for the upliftments of all other streams, which are yet to be taken up (less vulnerable). For example, assume that, at  $t=1$ , there remains only one upliftment of the most vulnerable stream  $i$  at  $t=9$  requiring 6 processing periods and only one upliftment of less vulnerable stream  $i'$  at  $t=5$  requiring 3 processing periods. That is, initially,  $h_{i1}=8$ ,  $d_{i1}=6$  and  $h_{i'1}=4$ ,  $d_{i'1}=3$ . We first take up stream  $i$ , and assign 6 processing periods to it. In that scenario, the upliftment of stream  $i'$  at  $t=5$  will have only 2 periods available for processing, i.e.,  $h_{i'1}$  becomes 2. However, if the upliftment of stream  $i$  required 4 processing periods or less, then after the assignment of stream  $i$ ,  $h_{i'1}$  will still be 4.

For every scenario that assigns  $\text{ceil}(d_{ik})$  processing period for an upliftment, we need to adjust the required processing period of the next upliftment, if any, of stream  $i$  as the extra amount beyond the requirement of the  $j^{\text{th}}$  upliftment will be available to meet the  $j+1^{\text{th}}$  upliftment of the same stream.

It is worth noting that we need not calculate the upliftment penalty heuristic afresh at time period  $t$  if the stream running on  $t-1$  period at  $U_3$  has not completed its minimum run length.

## 5.2 Heuristic for Spillage cost

Suppose we changeover to a stream  $i$  at time period  $t$  at unit  $U_1$ , i.e., some stream other than  $i$  has been running in  $U_1$  on and till time period  $t-1$ . Since every stream scheduled in a unit must run for the minimum run length  $b$ , stream  $i$ , once scheduled on time period  $t$  must run till time period  $t+b-1$ . Given the ullage positions of the input tanks of  $U_1$  at time period  $t$  and the fixed incoming rate of the raw materials, we can calculate the time periods when the other two streams will start spilling if not processed. The heuristic for spillage cost has two components. First, if a stream (or streams) starts spilling within time period  $t+b-1$ , we calculate the spillage amounts till that period and the corresponding spillage costs. This is the spillage cost that will definitely be incurred during this period. Next we check whether

more than one stream will start spilling in the period between  $t+b$  to  $t+2b-1$  (known as *closespill* in refinery parlor). If the answer is yes, then we would definitely be incurring spillage of at least one of the streams. To keep our heuristic admissible, we calculate the spillage amount of each stream during the period  $t+b$  to  $t+b-1$  and the corresponding cost. The lower of the two is added to the heuristic spillage cost.

If the stream scheduled on  $t$  is same as that scheduled on  $t-1$  and it has completed its minimum run length on or before  $t-1$ , closespill needs to be checked only during the period  $t+1$  to  $t+b$  since the unit is free to changeover to some other stream in the next time period. However, if the stream scheduled on  $t$  is yet to complete its minimum run length  $b$ , then its spillage cost remains the same for all the periods till it has run for  $b$  periods.

As mentioned earlier, every node in the search tree represents a partial schedule till time period  $t$ . This is obtained by extending the partial schedule till time period  $t-1$  of the parent node by feasible assignments of streams to the three units for the time period  $t$ , given the assignments till  $t-1$ . For every such node we update the cost incurred so far till period  $t$ , and calculate the heuristic upliftment penalty and spillage costs for the rest of the scheduling horizon. If the sum of the cost incurred and the heuristic costs is more than or equal to the currently best known cost of a complete solution, then the sub-tree rooted at this node need not be explored further.

### 5.3 Node Ordering

While extending a partial schedule till period  $t-1$  to period  $t$ , in which order should we try assigning the feasible streams to a unit in period  $t$ ? A pre-determined, static sequence of the streams may give rise to unnecessary search and may delay reaching the optimal solution. It would make better sense to look below the most potential assignment first which, hopefully, will lead us to a lower cost, and hence induce better pruning during the rest of the search. For example, consider assigning streams in unit  $U_3$ . If the most vulnerable stream has an upliftment for which the processing periods required is less than the available periods before the upliftment, it would be prudent to try out this stream first before other streams. For this purpose, before expanding a node – that is, before extending a partial schedule – we first dynamically order the feasible streams for a unit based on the information available at that point in time. We do this for unit  $U_1$  and unit  $U_3$  in the following manner.

For unit  $U_1$ , if the minimum run length for the stream  $i$  running in period  $t-1$  is not yet complete, then stream  $i$  is the only stream to be tried in period  $t$ . Let us consider the situation where stream  $i$  has completed its minimum run length at time period  $t-1$  or before. Let us assume that all the streams are feasible, that is, stream  $i$  has input stock and output ullage for one more time period and the other two streams have input stocks and output ullages for  $b$  periods. We first check whether the other two streams currently not being processed have a closespill in near future, that is, whether both the streams will start spilling within a gap of  $b$  periods if not processed in next  $2b$  periods. If there exists any such closespill, the stream that will spill first will be tried first followed by the other stream followed by stream  $i$ . If there is no possibility of closespill, we will first try stream  $i$  for one more period (to have longer stretches of runs) followed by the stream that will spill first followed by the remaining one.

While ordering nodes for unit  $U_3$ , we first calculate  $\hat{d}_i$ , the total number of processing periods required for each stream  $i$  to meet all the subsequent upliftments of the stream. If the summation of  $\hat{d}_i$ 's is more than the available processing periods  $h$ , we adjust the  $\hat{d}_i$ 's by reducing periods of processing of the streams starting from the least vulnerable stream. Since available processing periods  $h$  is an integer and the  $\hat{d}_i$ 's are, in general, not integers, we consider all possible combinations of floor and ceiling functions of reduced  $\hat{d}_i$ 's that add up to  $h$ , and pick up that combination that gives minimum upliftment failure penalty. This is necessary so that we do not waste one period of production to meet, say, remaining demand worth 0.1 period of processing of some stream. With only three streams, the combinations are not too many.

Given the adjusted integer  $\hat{d}_i^s$ , we order the streams for trial in the following manner. Streams with adjusted  $\hat{d}_i^s$  greater than zero having unmet demand in the next upliftment will be tried in order of decreasing vulnerability. Next to try are the streams with unmet demand in the next upliftment with no regard to their adjusted  $\hat{d}_i^s$ . If all streams have not been included, we next consider streams with  $\hat{d}_i^s$  greater than zero irrespective of their upliftment periods. If that too fails to put all the streams in the sequence, we next consider streams with  $\hat{d}_i^s$  equal to zero as well.

## 5.4 Initial Solution

The DFBB algorithm was fed with an initial solution which provided an upper bound on the optimal solution of the problem. The following is a broad overview of the deterministic algorithm that was used to obtain the initial solution. Conceptually, the algorithm considers unit  $U_1$  alone responsible for minimizing the spillage cost, unit  $U_3$  alone responsible for reducing the upliftment failure penalty, and unit  $U_2$  as a service unit trying to help out  $U_1$  and  $U_3$  in meeting their objectives. Each unit obeys the minimum run length restriction. The algorithm first calculates the minimum spillage that is definitely going to be wasted, and the minimum upliftment penalty that is definitely going to be incurred irrespective of the sequencing of the streams in the units. The input rates, the initial stocks at level 0 and the feed rates for  $U_1$  are used to get the spillage estimates, and the processing capacity of  $U_3$ , the initial stocks at level 3 and the upliftment schedule are used to estimate the minimal upliftment penalty.

The algorithm incrementally builds the schedule by assigning streams to units in successive time periods. For each unit, this decision is guided by a decision tree developed for the unit. At any given time period  $t$ ,  $U_1$  will be the first to decide the stream to be run at  $t$ . Since goal of  $U_1$  is to reduce spillage, it may find it necessary to schedule a stream which does not have sufficient output ullage for the purpose. In that case, it logs in a request to  $U_2$  to changeover to that stream, if possible. It also decides what it should run at  $t$ , if the request gets rejected. The next unit to choose its stream to be run at  $t$  is  $U_3$ .  $U_3$  decision tree is geared to meet the upliftment schedule. If  $U_3$  thinks it fit to run a stream for which it does not have enough input stock to sustain minimum run length, it can log in a request to  $U_2$  to changeover to that stream, if possible. Like  $U_1$ ,  $U_3$  also keeps an alternative ready in case the request gets rejected.  $U_2$  is the last to choose its stream for period  $t$ . It rejects the request(s) from  $U_1$  and/or  $U_3$ , in case the stream running in  $U_2$  at  $t-1$  is yet to complete its minimum run length. However, if it has no such restriction, it tries to changeover to the requested stream. If request has come from only one side, say  $U_1$ , and it can be honored (for example, there is enough ullage on the output side of  $U_2$  for the requested stream for the minimum run length),  $U_2$  does change over to the requested stream. However, if  $U_2$  has received requests at time  $t$  from both  $U_1$  and  $U_3$  to change over to different streams, it checks the spillage and upliftment penalty costs incurred till time period  $t$  and compares them with the corresponding least costs that have been calculated at the beginning. Based on the deviations,  $U_2$  decides to honor one of the requests and rejects the other. If there is no request from either side,  $U_2$  follows its own decision tree to choose the run for time period  $t$ .

## 6. COMPUTATIONAL EXPERIENCE

The proposed DFBB algorithm was coded in C++ and was run on 2.4 GHz Pentium 4 workstation with 1 GB RAM. The fixed data (Table 3) used for the test cases are real data collected from a refinery. The per unit spillage costs and the per unit upliftment failure penalties have been masked. However, their relative importances have been reflected in the values used. The capacity of each storage tank is assumed to be 10000 Mt for easy comparison of the input data for the cases considered.

Given the fixed refinery data in Table 3, the algorithm has been tried out in a total of 25 cases. For each case, the scheduling horizon was 30 time periods (days). The input of a case consists of the initial stock positions at all the four levels at the beginning of the scheduling horizon, the upliftment schedule, and information about the streams running in the units on the day just before the start of the scheduling horizon.

Table 3. Fixed Data

| Number of Units: 3 {U <sub>1</sub> : FEU; U <sub>2</sub> : SDU; U <sub>3</sub> : HFU} |                      |                 |     |       |                  |     |     |                  |                       |
|---------------------------------------------------------------------------------------|----------------------|-----------------|-----|-------|------------------|-----|-----|------------------|-----------------------|
| Number of Streams: 3 {1:IO; 2:HO; 3:DAO}                                              |                      |                 |     |       |                  |     |     |                  |                       |
| Minimum Run Length ( <i>b</i> ): 3                                                    |                      |                 |     |       |                  |     |     |                  |                       |
| Capacity (Mt): 10000 (For each stream in each level)                                  |                      |                 |     |       |                  |     |     |                  |                       |
| Scheduling Horizon( <i>H</i> ): 30                                                    |                      |                 |     |       |                  |     |     |                  |                       |
| Stream                                                                                | Input Rate<br>(MtPD) | Feed Rate(MtPD) |     |       | Yield Percentage |     |     | Spill<br>Penalty | Upliftment<br>Penalty |
|                                                                                       |                      | FEU             | SDU | HFU   | FEU              | SDU | HFU |                  |                       |
| 1                                                                                     | 620                  | 1250            | 780 | 546   | 60               | 70  | 99  | 10               | 14                    |
| 2                                                                                     | 255                  | 1030            | 576 | 403.2 | 50               | 70  | 99  | 12               | 12                    |
| 3                                                                                     | 310                  | 1050            | 521 | 364.7 | 65               | 70  | 99  | 14               | 10                    |

Table 4. Case data

#### Tank Stock Positions

| Streams | Stock Positions (Mt) |         |         |         |
|---------|----------------------|---------|---------|---------|
|         | Level 0              | Level 1 | Level 2 | Level 3 |
| 1       | 7000                 | 3000    | 2000    | 1000    |
| 2       | 9000                 | 5000    | 5000    | 1000    |
| 3       | 9500                 | 5000    | 5000    | 2000    |

#### Scheduled Upliftments (Mt)

| Day | Streams |      |      |
|-----|---------|------|------|
|     | 1       | 2    | 3    |
| 15  | 5000    | 3200 | 1500 |
| 28  | 6000    | 0    | 0    |

**Last Run:** (Unit:Stream:Run) : (U<sub>1</sub>:1:3); (U<sub>2</sub>:2:3); (U<sub>3</sub>:3:3)

To generate these test cases, we first designed a base case (Table 4). The inputs of the base case are chosen in a manner so as to require collaboration among the units to come up with the least cost schedule. A look at Table 4 data would reveal that unit  $U_3$  needs to process two streams (streams 1 and 2) to fulfill the upliftments, and the intermediate tank stocks for these streams are such that the  $U_3$  requirements for feed would necessitate processing of the corresponding streams in  $U_1$ . Again, the initial tank stocks at level 0 are chosen to create a situation for close spill. The Input Rate column (Table 3) and the Level 0 stocks (Table 4) considered together, show that streams 1, 2 and 3, if not processed in  $U_1$ , would start spilling on day 5, day 4 and day 2 respectively. It may be noted that the scheduled upliftment of stream 3 can be fully met from the stock available at level 3, and as such stream 3 does not require processing in  $U_1$  from upliftment point of view.

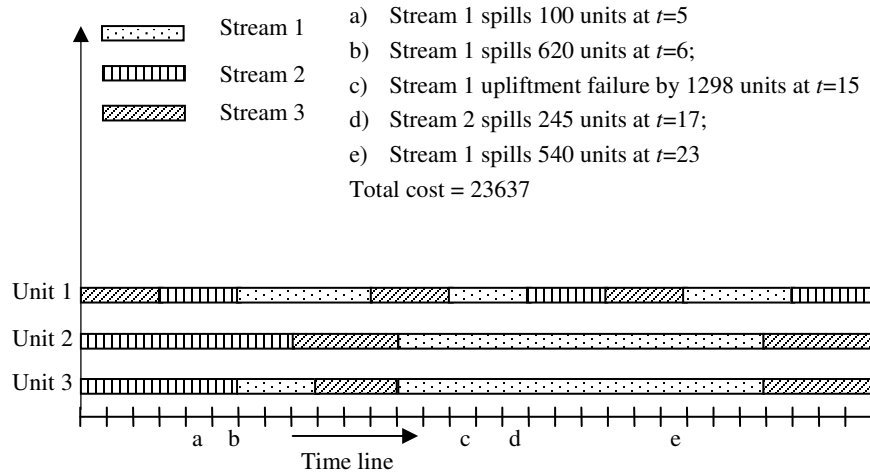


Figure 4. Gantt chart for the Initial Solution

Figure 4 is the Gantt chart of the Units' schedules as suggested by the initial solution while figure 5 shows the schedules as suggested by the DFBB scheme with solution values of 23637 and 5204 respectively. The initial solution acted in a greedy manner.  $U_1$  started with stream 3 to take care of the close spill, and in the process, spilled stream 1 on days 5 and 6.  $U_3$  was forced to run stream 3 on day 10 since there was not enough input stock for the other streams. The improvement of 78% over the initial solution was achieved by the DFBB in about 8 minutes after expanding 6.3 million nodes with 10 updates. The optimal solution was found after expanding 5 million nodes but the algorithm needed to expand 1.3 million more nodes before concluding the value to be the optimal one. The algorithm solves the problem by judicious allocation of days to streams in units. It continued processing stream 1 in  $U_1$  on the first day to create cushion for delaying its spillage and also to meet the requirement of stream 1 in  $U_3$ , switched to stream 3 on day 2 to avoid its spillage, allowed stream 2 to spill for some time till it could switch over to it on day 5. The solution manages to strike a balance between spillage and upliftment failure by quick changeovers in  $U_1$  in a sequence that helps  $U_3$  to fulfill its upliftment schedule to the extent possible.

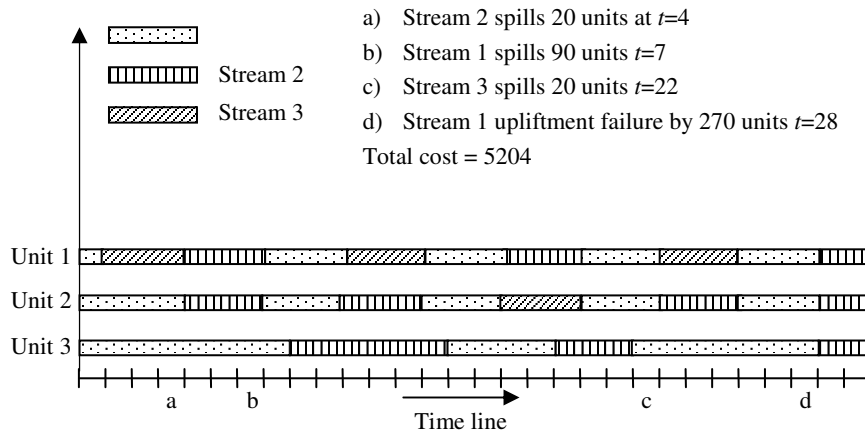


Figure 5. Gantt chart for the DFBB solution

Table 5 gives the details of the 25 case instances that include the base case as Case 1. These cases were generated by varying the input parameters of the base case. Each case was obtained by altering the initial stock positions and/or the streams to be uplifted along with their amounts. Table 6 summarizes our experiences with the DFBB scheme when applied to these 25 cases. The algorithm was run on each case with a maximum time limit of one hour. The initial and the DFBB solutions have been rounded to nearest integers. The initial solution was optimal in three cases with a cost of 0. The DFBB algorithm could output optimal solutions in 17 of the remaining 22 cases. In each of the five cases where the algorithm could not terminate in the stipulated one hour, the last update was a substantial improvement over the initial one.

Table 5. Case Instances

| Case No. | Initial Stock Position ('00 Mt) |    |    |    | Scheduled Upliftments ('00 Mt) |          |          |          |
|----------|---------------------------------|----|----|----|--------------------------------|----------|----------|----------|
|          | Levels                          |    |    |    | Day                            | Streams  |          |          |
|          | 0                               | 1  | 2  | 3  |                                | 1        | 2        | 3        |
| 1        | 70                              | 30 | 20 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0  |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 2        | 70                              | 30 | 20 | 10 | 15<br>28                       | 32<br>60 | 50<br>0  | 15<br>0  |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 3        | 70                              | 30 | 20 | 10 | 15<br>28                       | 32<br>60 | 15<br>0  | 50<br>0  |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 4        | 70                              | 30 | 20 | 10 | 15<br>28                       | 15<br>60 | 32<br>0  | 50<br>0  |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 5        | 70                              | 30 | 20 | 10 | 15<br>28                       | 15<br>60 | 50<br>0  | 32<br>0  |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 6        | 70                              | 30 | 20 | 10 | 15<br>28                       | 15<br>0  | 50<br>0  | 32<br>60 |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 7        | 70                              | 30 | 20 | 10 | 15<br>28                       | 15<br>0  | 50<br>60 | 32<br>0  |
|          | 90                              | 50 | 50 | 10 |                                |          |          |          |
|          | 95                              | 50 | 50 | 20 |                                |          |          |          |
| 8        | 50                              | 30 | 20 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0  |
|          | 70                              | 50 | 50 | 10 |                                |          |          |          |
|          | 65                              | 50 | 50 | 20 |                                |          |          |          |
| 9        | 50                              | 60 | 20 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0  |
|          | 70                              | 5  | 50 | 10 |                                |          |          |          |
|          | 65                              | 20 | 50 | 20 |                                |          |          |          |
| 10       | 50                              | 60 | 40 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0  |
|          | 70                              | 50 | 10 | 10 |                                |          |          |          |
|          | 65                              | 20 | 20 | 20 |                                |          |          |          |
| 11       | 50                              | 50 | 50 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0  |
|          | 50                              | 50 | 50 | 10 |                                |          |          |          |
|          | 50                              | 50 | 50 | 20 |                                |          |          |          |
| 12       | 50                              | 50 | 50 | 10 | 15<br>28                       | 32<br>60 | 15<br>0  | 50<br>0  |
|          | 50                              | 50 | 50 | 10 |                                |          |          |          |
|          | 50                              | 50 | 50 | 20 |                                |          |          |          |
| 13       | 50                              | 50 | 50 | 10 | 15<br>28                       | 32<br>0  | 15<br>0  | 55<br>60 |
|          | 50                              | 50 | 50 | 10 |                                |          |          |          |
|          | 50                              | 50 | 50 | 20 |                                |          |          |          |

| Case No. | Initial Stock Position ('00 Mt) |    |    |    | Scheduled Upliftments ('00 Mt) |          |          |         |
|----------|---------------------------------|----|----|----|--------------------------------|----------|----------|---------|
|          | Levels                          |    |    |    | Day                            | Streams  |          |         |
|          | 0                               | 1  | 2  | 3  |                                | 1        | 2        | 3       |
| 14       | 50                              | 50 | 50 | 10 | 15<br>28                       | 32<br>0  | 15<br>55 | 0<br>60 |
|          | 50                              | 50 | 50 | 10 |                                |          |          |         |
|          | 50                              | 50 | 50 | 20 |                                |          |          |         |
| 15       | 50                              | 70 | 50 | 10 | 15<br>28                       | 32<br>0  | 15<br>55 | 0<br>60 |
|          | 90                              | 50 | 50 | 10 |                                |          |          |         |
|          | 55                              | 50 | 80 | 20 |                                |          |          |         |
| 16       | 70                              | 30 | 20 | 10 | 15<br>28                       | 32<br>0  | 15<br>55 | 0<br>60 |
|          | 90                              | 50 | 50 | 10 |                                |          |          |         |
|          | 95                              | 50 | 50 | 20 |                                |          |          |         |
| 17       | 70                              | 80 | 40 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 70 | 50 | 10 |                                |          |          |         |
|          | 95                              | 60 | 80 | 20 |                                |          |          |         |
| 18       | 70                              | 80 | 40 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 20                              | 70 | 50 | 10 |                                |          |          |         |
|          | 95                              | 60 | 80 | 20 |                                |          |          |         |
| 19       | 70                              | 80 | 40 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 70 | 50 | 10 |                                |          |          |         |
|          | 15                              | 60 | 80 | 20 |                                |          |          |         |
| 20       | 70                              | 80 | 40 | 20 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 70 | 50 | 10 |                                |          |          |         |
|          | 85                              | 20 | 10 | 0  |                                |          |          |         |
| 21       | 87                              | 10 | 10 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 10 | 10 | 10 |                                |          |          |         |
|          | 95                              | 10 | 10 | 20 |                                |          |          |         |
| 22       | 60                              | 10 | 10 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 10 | 10 | 10 |                                |          |          |         |
|          | 95                              | 10 | 10 | 20 |                                |          |          |         |
| 23       | 50                              | 10 | 10 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 50                              | 10 | 10 | 10 |                                |          |          |         |
|          | 50                              | 10 | 10 | 20 |                                |          |          |         |
| 24       | 50                              | 10 | 10 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 10 | 10 | 10 |                                |          |          |         |
|          | 95                              | 10 | 10 | 20 |                                |          |          |         |
| 25       | 50                              | 20 | 10 | 10 | 15<br>28                       | 50<br>60 | 32<br>0  | 15<br>0 |
|          | 90                              | 10 | 10 | 10 |                                |          |          |         |
|          | 95                              | 10 | 10 | 20 |                                |          |          |         |



Table 6. Case Results

| Case No. | Initial Solution | DFBB Solution | Time                  | Number of Updates | Total nodes expanded | Node count at last update |
|----------|------------------|---------------|-----------------------|-------------------|----------------------|---------------------------|
| 1        | 23637            | 5204          | 7m59.3s               | 10                | 6306816              | 5095442                   |
| 2        | 23637            | 2050          | 17.9s                 | 11                | 139304               | 19925                     |
| 3        | 31205            | 3746          | 0.14s                 | 14                | 1392                 | 1380                      |
| 4        | 33314            | 6146          | 0.2s                  | 13                | 826                  | 814                       |
| 5        | 30964            | 2688          | 0.2s                  | 11                | 1143                 | 1093                      |
| 6        | 33261            | 19141         | 49.7s                 | 7                 | 66905                | 18504                     |
| 7        | 32370            | 20408*        | 9m20.5s <sup>†</sup>  | 4                 | DNT                  | 882 <sup>†</sup>          |
| 8        | 18162            | 0             | 0.01s                 | 2                 | 37                   | 37                        |
| 9        | 18919            | 0             | 0.01s                 | 1                 | 31                   | 31                        |
| 10       | 0                | 0             | 0.01s                 | 0                 | 1                    | 0                         |
| 11       | 0                | 0             | 0.01s                 | 0                 | 1                    | 0                         |
| 12       | 34632            | 5002          | 6.1s                  | 5                 | 75882                | 146                       |
| 13       | 53811            | 18488         | 5m20.5s               | 10                | 2778764              | 343                       |
| 14       | 35500            | 22097*        | 42m31.1s <sup>†</sup> | 4                 | DNT                  | 19687292 <sup>†</sup>     |
| 15       | 24100            | 22097*        | 9m53.3s <sup>†</sup>  | 1                 | DNT                  | 31 <sup>†</sup>           |
| 16       | 44430            | 18995*        | 10m19.6s <sup>†</sup> | 14                | DNT                  | 223951 <sup>†</sup>       |
| 17       | 50551            | 1420          | 12.26s                | 28                | 110979               | 7844                      |
| 18       | 23384            | 0             | 2.50s                 | 18                | 139714               | 139714                    |
| 19       | 0                | 0             | 0.01s                 | 0                 | 1                    | 0                         |
| 20       | 25297            | 11758*        | 5m35.6s <sup>†</sup>  | 6                 | DNT                  | 951 <sup>†</sup>          |
| 21       | 70109            | 52027         | 19.34s                | 8                 | 462315               | 456489                    |
| 22       | 69864            | 39407         | 28.2s                 | 13                | 684973               | 167125                    |
| 23       | 51639            | 3784          | 4.07s                 | 14                | 95494                | 91218                     |
| 24       | 71892            | 39407         | 57.2s                 | 19                | 1438665              | 353513                    |
| 25       | 71892            | 24271         | 19.4s                 | 12                | 433665               | 84929                     |

\*DFBB solution at the last update count.

<sup>†</sup>Time and Total nodes expanded till the last update.

DNT: Did not terminate in 1 hour.

## 7. CONCLUDING REMARKS

In this paper we have suggested a heuristic scheme for scheduling cascaded blocked-out continuous processing units with simultaneous arrivals of the raw materials and multiple upliftment due dates. The scheme has been found to work well under varying input situations. However, the planners may not feel very comfortable with the 'optimal' solutions being obtained from the suggested algorithm. The scheme, in its present state, solves a simplified version of the original problem since it does not include changeover cost in its objective function. There is no assurance that the schedule suggested by the scheme would still remain optimal when changeover cost is factored in. As mentioned earlier we had to leave out changeover cost from this version of the algorithm since, given a partial schedule, we are yet to come up with a 'good' admissible estimate for the changeover cost over the remaining scheduling horizon. Work is currently in progress in this regard. Once this heuristic is ready, we expect the complete DFBB scheme to be able to give optimal solutions for most of the real life problems in reasonable time.

**REFERENCE:**

- Alle, A and Pinto, J. M., "A mathematical programming approach for cyclic production and cleaning scheduling of multistage continuous plants," *Computers and Chemical Engineering*, 28(2004), 3–15.
- Blömer, F. and Günther, H.O., "Scheduling of a Multi-Product Batch Process in the Chemical Industry," *Computers in Industry*, 36(1998), 245-259.
- Frauendorfer, K. and Königspurger, E., "Concepts for improving scheduling decisions: An application in the chemical industry," *International Journal of Production Economics*, 46-47, (1996), 27-38
- Garey, M. R. and Johnson, D. S., "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H Freeman and Company, (1979).
- Graham, R.L., Lawler, E.L., Lenstra, J.K. and Kan, A.H.G.R., "Optimization and approximation in deterministic sequencing and scheduling: A survey", *Ann. Discrete Mathematics*-5(1979), 287-326.
- Grossmann I.E., Quesada J., Raman R. and Voudouris V., "Mixed integer optimization techniques for the design and scheduling of batch processes," *Batch Processing Systems Engineering*, eds. G.V. Reklaitis, A.K. Sunol, D.W.T. Rippin and O. Hortacsu, Springer, Berlin(1996), 451-494.
- Ierapetritou, M.G. and Floudas, C.A., "Effective Continuous-Time Formulation for Short-Term Scheduling.-2. Continuous and Semicontinuous Processes," *Ind. Eng. Chem. Res.* , 37, (1998), 4360-4374.
- Ierapetritou, M.G. Hene, T.S. and Floudas, C.A., "Effective Continuous-Time Formulation for Short-Term Scheduling.-3. Multiple Intermediate Due Dates," *Ind. Eng. Chem. Res.* , 38, (1999), 3446-3461.
- Jain, V. and Grossmann, I.E., "Cyclic scheduling of continuous parallel-process units with decaying performance," *AIChE Journal*, 44(1998), 1623.
- Jia, Z. and Ierapetritou, M.G., "Efficient short term scheduling of refinery operations based on continuous time formulation," *Computers and Chemical Engineering*, 28(2004), 1001–1019.
- Kondili, E., C.C. Pantelides and R.W.H. Sargent, "A General Algorithm for Short-Term Scheduling of Batch Operations - 1. Mixed Integer Linear Programming Formulation," *Computers and Chemical Engineering*, 17,(1993), 211-227
- Kudva, Elkamel, Pekny and Reklaitis, G.V., "Heuristic Algorithm for Scheduling Batch and Semicontinuous Plants with Production Deadlines, Intermediate Storage Limitations and Equipment Changeover Costs," *Computers and Chemical Engineering*, 18,(1994), 859-875.
- Ku and Karimi, "Completion Time Algorithms for Serial Multiproduct Batch Processes with Shared Storage," *Computers and Chemical Engineering*, 14, (1990), 49-69
- Ku and Karimi, "Scheduling algorithm for serial multiproduct batch processes with tardiness penalties," *Computers and Chemical Engineering*, 15,(1991), 283-286
- Mendez, C. and Cerda, J., "An MILP continuous-time formulation for short-term scheduling of multiproduct continuous facilities," *Computers and Chemical Engineering*, 26,(2002), 687-695
- Pinto, J.M. and Grossmann, I.E., "Assignment and Sequencing models for the scheduling of process systems," *Annals of Operations Research*, 81(1998), 433-466.
- Rippin, D.W.T., "Batch process systems engineering: a retrospective and prospective review," *Computers and Chemical Engineering*, 17 Suppl.(1993), S1 – S13.
- Reklaitis, G.V., "Overview of the Planning and Scheduling technologies," *Latin American Applied Research*(2000)

- Wiede and Reklaitis, G.V., "Determination of completion times for serial multiproduct processes-2. A multiunit finite intermediate storage system," *Computers and Chemical Engineering*, 11,(1987a),345-356
- Wiede and Reklaitis, G.V., "Determination of completion times for serial multiproduct processes-3. Mixed intermediate storage systems," *Computers and Chemical Engineering*,11,(1987b),357-368
- Sargent R.W.H., "Introduction:25 years of progress in process systems engineering," *Computers and Chemical Engineering*,28,(2004),437-439
- Schilling, G. and Pantelides, C.C., "A simple continuous time process scheduling formulation and a novel solution algorithm," *Computers and Chemical Engineering*, 20,(1996),S1221-S1226
- Shah, N., "Single-and multisite planning and scheduling: current status and future challenges," *AIChE Symposium Series*(1998).
- Shah, N., Pantelides, C.C. and Sargent, R.W.H., "A General Algorithm for Short-Term Scheduling of Batch Operations-2. Computational Issues," *Computers and Chemical Engineering*,17,(1993),229-244
- Zhang, X. ad Sargent, R.W.H., "The optimal operation of mixed production facilities-A general formulation and some approaches for the solution," *Computers and Chemical Engineering*, 20,(1996) 897-904.

# **Production Scheduling**

Full Papers

# A BI-CRITERIA APPROACH TO SCHEDULING WAFER FABRICATION FACILITIES

Michele Pfund<sup>1</sup>, Hari Balasubramanian<sup>1</sup>, John Fowler, and Scott Mason<sup>2</sup>

1: Industrial Engineering, Arizona State University, 2:Industrial Engineering, University of Arkansas

**Abstract:** In this paper, we propose the use of the desirability function in the Shifting Bottleneck (SB) heuristic for complex job shops to enable the of both makespan and total weighted tardiness. We implement the desirability function both at the toolgroup or sub-problem procedure (SSP) level and at the machine criticality level (MCM) of the SB heuristic. We test our approaches on the "Minifab" model by El Adl et al. (1996), the most succinct representation of a wafer fab in the open literature. Given the decision makers weights on the two objectives, the results show that the desirability-based approach leads to solutions better than those generated by the SB-based total weighted tardiness approach and commonly used dispatching rules.

**Key words:** Bicriteria, Shifting Bottleneck, Complex Job Shop

## 1. INTRODUCTION

In semiconductor wafer fabrication facilities, there often are dozens of process flows and each process flow contains 200-500 processing steps and requires more than 100 machines. These machines are expensive, ranging in price from \$50,000 to over \$10 million per tool. It is common for groups of identical machines to process lots in parallel, thereby forming a *toolgroup*. A given part may visit a toolgroup many times as part of its process flow. The main consequence of this feature of wafer fabs – called *re-entrant flow* – is that wafers at different stages in their manufacturing cycle have to compete with each other for the same machines. The manner in which this competition is resolved has a clear impact on fab objectives. The manufacturing environment, therefore, is different in several ways from both traditional flow shops as well as job shops.

Additionally, the nature and duration of the various operations in a fab process flow are quite diverse. While some operations require 30 minutes or less to process a *lot* of 25 wafers, others may require over twelve hours. Many of these long operations involve batch processing of lots. Batch processing machines tend to process multiple lots (1-6) at a time, causing the formation of long queues in front of these serial non-batch tools and ultimately a non-linear flow of products in the factory. In reality, it is not uncommon for one-third of all fab operations to involve batch processing. There are also some fab machines, such as ion implanters, that require significant sequence-dependent setups. If not scheduled appropriately, these machines can become fab bottlenecks.

The primary goal of this research effort is to develop solution methodologies that provide good utilization and customer satisfaction performance for semiconductor wafer fabs. Thus we use makespan and total weighted tardiness (TWT) as our performance measures. Wafer fabs are modeled as complex job shops by Mason *et al.* (2002), who extend the classical job shop work of Pinedo and Singer (1999) to

develop a disjunctive graph formulation and modified Shifting Bottleneck heuristic (MSBH) for the wafer fab scheduling problem  $FJm | r_j, s_{jk}, p\text{-batch}, recrc | \sum w_j T_j$ . Unlike most previous job shop scheduling efforts, Mason *et al.* (2002) account for batch processing tools (*p-batch*), different arrival times of job ( $r_j$ ), toolgroups (which is why the environment is a flexible job shop represented as *FJm*), sequence-dependent setups ( $s_{jk}$ ), and recirculating product flow (*recrc*), all of them key features that characterize manufacturing in wafer fabs.

We consider bicriteria problem involving makespan and TWT in the complex job shop environment; our main goal in this research to extend the work of Mason *et al.* (2002) to include multiple criteria. We note here that even single criterion problems in the environment are NP-hard. Therefore our approach in the paper is to develop a computationally efficient heuristic that is able to accommodate multiple criteria.

## 2. RELATED WORK

Multicriteria scheduling research arose from the need to address real world scheduling problems, which seldom have a single objective function. A schedule that is good for one objective function may in fact be quite poor for another. Decision makers must carefully evaluate the trade-offs involved in considering several different criteria in practical scheduling applications.

Multicriteria problems can be considered in many ways. Consider for example a bicriteria scheduling problem with objectives of minimizing  $\gamma_1$  and  $\gamma_2$ . Sometimes the decision-maker has *a priori* information regarding the nature of optimization to be performed. For instance, criteria  $\gamma_1$  may be of primary importance and  $\gamma_2$  of secondary interest. In other words, a solution best in  $\gamma_2$  may be desired among all solutions that are best in  $\gamma_1$ . This is called hierarchical or lexicographic optimisation. The decision-maker may also have composite linear function of the form  $F_l(\gamma_1, \gamma_2) = \alpha\gamma_1 + (1-\alpha)\gamma_2$ ,  $0 \leq \alpha \leq 1$  in mind that needs to be minimized. The weighted sum translates multiple objectives into a single objective value for a proposed schedule. In this way, alternative schedules can be compared easily using only a single objective or fitness value. Some difficulty can be experienced in setting the weighting factors in practice, primarily due to the dimensionality of the objective function criteria in practice (Van Wassenhove and Baker 1982). Care must be taken to ensure the scale of  $\gamma_1$  (e.g.,  $\sum w_j C_j$ ) does not dominate  $\gamma_2$  (e.g.,  $C_{\max}$ ). Recently, Kacem *et al.* (2002) introduced a homogenization approach for dealing with this potential scale difference. Cochran *et al.* (2003) discuss several schemes for weighting the objectives.

Another technique is to generate the set of pareto optimal or non-dominated points for the decision-maker to choose from. A schedule is called pareto optimal or non-dominated if there exists no other schedule that has values of  $\gamma_1$  and  $\gamma_2$  such that at least one of the objectives is lower. The decision maker can now choose from this set of non-dominated solutions, the schedule that works best. This approach is called the *a posteriori* approach and also is the most difficult.

For comprehensive surveys on multicriteria scheduling we refer to Foote *et al.* (1988), Nagar *et al.* (1995), T'kindt and Billaut (2002) and for a more recent study to Hoogeveen *et al.* (2004). Of other papers, we refer here only to the papers that involve the job shop environment. Esquivel *et al.* (2002) and Kacem *et al.* (2002) investigate the generation of efficient schedules in classical and flexible job shops. Lima *et al.* (1999) and Itoh *et al.* (1993) consider the minimization a function that involves all the objectives under study in classical job shops. Recently Petrovic and Fayad (2004) consider minimization of makespan in real-world job shops using the shifting bottleneck procedure hybridised with genetic algorithms. They propose the extension of this work to multicriteria problems.

In this paper, we consider bicriteria scheduling problem in which we combine  $C_{\max}$  and TWT into a single aggregation function. Our aggregation function, however, is different from the linear combination of objectives described earlier. We use, instead, the desirability function to aggregate the objectives. We assume the decision maker has a prefixed goal/target value and an upper/worst-case value for every criterion. We also assume the decision-maker, just as in the linear combination case, is aware of the priorities on the two objectives. Our research is new in two respects. Except for the proposition of Petrovic and Fayad (2004) the shifting bottleneck procedure has been used to minimize only single criteria objective functions, such as  $C_{\max}$  (Adams *et al.* 1988),  $L_{\max}$  (Ovacik and Uzsoy 1997), and TWT (Pinedo and Singer 1999) for classical job shops. Further, the complex job shop environment has only been studied for TWT (Mason *et al.* 2002). Here we present a solution methodology for the bicriteria optimization makespan and TWT in a complex job shop.

### 3. THE DESIRABILITY FUNCTION

Derringer and Suich (1980) originally suggested the desirability function approach to optimising multiple responses of interest (the term “response” here is the same as “criterion” or “objective”). Each response  $y_i$  is converted into an individual desirability function  $\delta_i$  that varies over the range zero to one. If  $y_i$  is outside the acceptable range defined by the user, then  $\delta_i = 0$  ; if  $y_i$  meets the goal, then  $\delta_i = 1$ .

Both objectives (makespan and TWT) in our research are to be minimized. Let  $U$  be the maximum allowable value for the response  $y_i$  and let  $G$  be the goal value for  $y_i$ . When minimizing the response  $y_i$ ,  $\delta_i = 1$  if  $y_i < G$ . Further, if  $G \leq y_i \leq U$ ,

$$\delta_i = \left( \frac{U - y_i}{U - G} \right)^{\alpha_i} \quad (1)$$

Otherwise, if  $y_i > U$ , then  $\delta_i = 0$ . In the above equation  $\alpha_i$  is a real valued number that is called the weight on the desirability function. When all  $\alpha_i=1$ , the desirability function is linear. Choosing  $\alpha_i > 1$  places more emphasis on being close to the target value while choosing  $0 < \alpha_i < 1$  makes this less important.

Once the individual desirabilities have been calculated, the overall desirability  $D$ , to be maximized, is the geometric mean of the individual desirabilities and is computed as:

$$D = (\delta_1 \delta_2 \dots \delta_m)^{1/m} \quad (2)$$

In (2),  $m$  is the total number of responses. For our research,  $m = 2$ . We represent the desirabilities of makespan and TWT as  $\delta_{cmax}$ , and  $\delta_{twt}$ . Each desirability  $\delta_i$  in our experiments will have goal value  $G_i$  and maximum (upper) limit  $U_i \quad \forall i \in \{cmax, twt\}$ .

## 4. METHODOLOGY

For the  $FJm | r_j, s_{jk}, p - batch, recrc | \sum w_j T_j$  problem the following shifting bottleneck algorithm is used for obtaining a schedule (Mason *et al.* (2002)):

1. Let  $M$  denote the set of all  $m$  tool groups. Initially, the set of tool groups that have been sequenced or scheduled,  $M_o$ , is empty.
2. Identify and solve the subproblems for each tool group  $i \in M \setminus M_o$ . **(SSP level)**
3. Identify a critical or bottleneck tool group  $k \in M \setminus M_o$ . **(MCM Level)**
4. Sequence tool group  $k$  using the subproblem solution from Step 2. Set  $M_o = M_o \cup \{k\}$ .
5. Optional: Re-optimize the schedule for each tool group  $m \in M_o$ , considering the newly added disjunctive arcs for tool group  $k$ .
6. If  $M = M_o$ , stop. Otherwise, go to Step 2.

We refer to step 2 as the SSP level and step 3 as the MCM level. At the SSP level, the toolgroups are scheduled (using a heuristic), and an objective function evaluation is obtained after the schedule is inserted into the graph. In the MCM level (step 3), this objective function is then used to determine the toolgroup that is the most critical toolgroup. In the algorithm shown above total weighted tardiness is used as the objective function.

At the SSP and MCM levels, we propose to use the desirability function of Derringer and Suich (1980), which seeks to optimize an aggregated function of several different objectives. We combine makespan and total weighted tardiness into a single desirability function to be maximized.

### 4.1 SSP Level

We use the Apparent Tardiness Cost with Setups and Ready Times (ATCSR) heuristic of Gadkari (2003) to schedule jobs on each toolgroup at the SSP level:

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - \max(r_j, t), 0)}{k_1 \bar{p}}\right) \exp\left(-\frac{s_{lj}}{k_2 \bar{s}}\right) \exp\left(-\frac{\max(r_j - t, 0)}{k_3 \bar{p}}\right)$$

In the above equation,  $d_j$  is the operational due-date of job  $j$  at the toolgroup (SSP) level,  $w_j$  is job  $j$ 's weight or priority, and  $r_j$  is the ready time of job  $j$  with  $d_j$ . The  $r_j$  value is set by finding the critical path of the disjunctive graph. The setup time incurred when changing from job  $l$  to job  $j$  is denoted by  $s_{lj}$ , the average processing time of all remaining unscheduled jobs by  $\bar{p}$ , and  $\bar{s}$  is the average setup time. The three smoothing parameters  $k_1$ ,  $k_2$ , and  $k_3$  determine the relative importance of the exponential terms in relation to each other. The heuristic works as follows: we first identify the machine  $j$  that is available to process jobs the earliest ( $t$  denotes the time at which the machine is available). Next, we calculate the  $I_j(t)$  value for all unscheduled jobs and schedule, at time  $t$  on machine  $j$ , the job that has the highest  $I_j(t)$  value.

At the SSP level, we vary the parameters  $k_1$ ,  $k_2$  and  $k_3$  using a grid search to generate schedules with differing values of the two performance measures. Let  $I$  represent the total number of  $k_1, k_2$  and  $k_3$  smoothing parameter combinations evaluated over the grid space at the SSP level. The parameters are set based on a combination of pilot runs and using some guidelines from Gadkari (2003). Schedule  $i$  ( $i = 1 \dots I$ ) is characterized by its corresponding objective function values for the two performance measures of interest,  $Cmax^{(i)}$  and  $TWT^{(i)}$ . Further, let  $Cmax_{(min)}$  ( $Cmax_{(max)}$ ) and  $TWT_{(min)}$  ( $TWT_{(max)}$ ) denote the corresponding grid space's minimum (maximum) objective function values over



all  $l$  schedules evaluated at the SSP level. Let  $D_{(max)} = \max_{i=1..l} D^{(i)}$ , where  $D^{(i)}$  denotes the combined desirability of schedule  $i$ . We identify the schedule corresponding to  $D_{(max)}$  as  $S^*$ . Finally, let  $\alpha_{Cmax}$  and  $\alpha_{TWT}$  signify the desirability weights for makespan and TWT, respectively.

Procedure **SetGoals** below determines the upper and goal values for each objective of interest by generating  $l$  schedules over the smoothing parameter grid space for the toolgroup currently under study in the SSP. Then, Procedure **FindMostDesirable** identifies the most desirable schedule within the set of  $l$  generated schedules.

**Procedure SetGoals**

For  $i = 1$  to  $l$

    Create schedule  $i$  using the  $i$ th combination of smoothing parameters  $k_1, k_2$  and  $k_3$  on the subproblem.

    Record  $Cmax^{(i)}$ ,  $TWT^{(i)}$ .

    If  $\alpha^{(i)} < \alpha_{(min)}$  Then  $\alpha_{(min)} = \alpha^{(i)} \quad \forall \alpha \in \{Cmax, TWT\}$

    If  $\alpha^{(i)} > \alpha_{(max)}$  Then  $\alpha_{(max)} = \alpha^{(i)} \quad \forall \alpha \in \{Cmax, TWT\}$

Next  $i$

Goal value  $G_\alpha = \alpha_{(min)} \quad \forall \alpha \in \{Cmax, TWT\}$

Upper value  $U_\alpha = \alpha_{(max)} \quad \forall \alpha \in \{Cmax, TWT\}$

**Procedure FindMostDesirable**

$D_{(max)} = 0$

For  $i = 1$  to  $l$

    Calculate  $D^{(i)}$  using Equations (1) and (2)

    If  $D^{(i)} > D_{(max)}$  Then

$D_{(max)} = D^{(i)}$

$S^* = i$

    End If

Next  $i$

Thus, the goal (upper) limit is thus set to the best (worst) value observed for each objective over the  $l$  different schedules considered at the SSP level.

## 4.2 MCM level

Before the desirability function can be used at the MCM level, the upper limits and goal values for each of the three objectives of interest must be determined. We employ a procedure similar to the one used at the SSP level. The only difference is that at the MCM level, we are interested in identifying and scheduling the toolgroup with the least (lowest) combined desirability value. We consider two different approaches for identifying the critical toolgroup at the MCM level. First, each toolgroup schedule's impact on the makespan, cycle time and TWT of the entire complex job shop is assessed when identifying the critical toolgroup ("Global MCM") by inserting the schedule of the toolgroups into the disjunctive graph. Alternatively, the critical toolgroup can be identified using only SSP level performance metrics

(“Local MCM”) (i.e., do not consider the toolgroup’s impact on the rest of the complex job shop). Regardless of which MCM approach is used, the end result of this step of the MSBH is the scheduling of the critical toolgroup via the insertion of arcs into the corresponding problem’s disjunctive graph.

## 5. TESTING AND EXPERIMENTATION

Table 1 shows the different combination of approaches at the SSP and MCM levels that we investigate in this paper. Approach 1 is the MSBH of Mason *et al.* (2002) that seeks to minimize TWT. The other five approaches use some combination of desirability (“Des”) and TWT minimization. We employ the naming convention “SSP- $\gamma$ \_MCM- $\omega$ ” to describe an approach that uses  $\gamma \in \{\text{TWT}, \text{Des}\}$  at the SSP level and  $\omega \in \{\text{TWT}, \text{Des}(\text{Local}), \text{Des}(\text{Global})\}$  at the MCM level. For example, Approach 6 in Table 1 corresponds to SSP-Des\_MCM-Des(Global). We also compare the six approaches in Table 1 with the following pure dispatching-based approaches: Critical Ratio (CR), Earliest Due Date (EDD) and First In First Out (FIFO).

### 5.1 Experimental Testbed

We test our approaches on the “Minifab” model of El Adl *et al.* (1996), perhaps the most succinct representation of a wafer fab in the open literature. The Minifab (figure 1) consists of three toolgroups and essential features that characterize wafer fabs: batch processing machines (Toolgroup 1) with maximum batch size of three jobs, (e.g., a diffusion oven); identical serial processing machines operating in parallel (e.g., a photolithography stepper), while Toolgroup 3 consists of a single machine characterized by sequence-dependent setups (e.g., an ion implanter).

We first consider 20-job static instances of the Minifab wherein  $r_j = 0 \quad \forall j$ . There are two part types in this model: 10 jobs of Part Type A and 10 jobs of Part Type B. The weights (priorities) of the jobs, regardless of part type, are generated using a discrete uniform distribution  $w_j \sim DU[1,100]$ . The due date for job  $j$  is assigned using the notion of a flow factor  $f_j$  used to represent some multiple of job  $j$ ’s theoretical (raw) processing time  $RPT_j$ :  $d_j = r_j + f_j RPT_j$ , where  $r_j = 0$  in the minifab instances. In order to generate reasonable values for  $f_j$ , we examine a single static instance using FIFO dispatching at all toolgroups. From each job’s completion time  $C_j$  we estimate  $f_j = (C_j - r_j) / RPT_j$ . Let  $f_{\max} = \max_j f_j$  and  $f_{\min} = \min_j f_j$ . Since FIFO is independent of due-date and weight settings, only one instance is necessary to obtain  $f_{\max}$  and  $f_{\min}$  values. For our experiments involving the MSBH, we generate each flow factor  $f_j$  uniformly over the range  $[f_{\min}, f_{\min} + f_{\max} / 2]$ . We generate 20 different experimental instances of the Minifab model, each with their own unique flow factor and job weight values, using common random numbers (Law and Kelton 2000) across the scheduling approaches to better discriminate between their performance.

### 5.2 Desirability Function Weights

Previously, we discussed the procedure for setting the upper and goal limits for the three different objectives at the MCM and SSP levels. However, we still must determine how desirability weights  $\alpha_i$

will be set for  $i \in (C_{max}, TWT)$ . Since the three weights are blended together into a single objective function, their ratios relative to each other are important. Experimentation with these weights involves testing the weight values between zero and one, subject to the constraint that  $\sum_i \alpha_i = 1$  (Dabbas *et al.*

2003), i.e. this is a mixture experiment. Table 2 describes the desirability weight settings used in our experimentation. Note that the second combination of weights in the table represents the typical TWT optimization associated with the MSBH of Mason *et al.* (2002). These weight setting combinations are obtained from an augmented simplex centroid design of 2 variables in a mixture experiment (Myers and Montgomery 1995).

In the presentation of our results for the two models, we assume that the decision maker has placed equal emphasis on both  $C_{max}$  and TWT (i.e.,  $\alpha_{C_{max}} = \alpha_{TWT} = 0.5$ ) at the output level, for the final job-shop schedule. However, as the desirability function used to choose either the schedule at the SSP level or the critical toolgroup at the MCM level is just a heuristic procedure, it is necessary to test a number of possible combinations of weights at the SSP and MCM levels (other than the equal emphasis ones) to get a final solution that is equally good in  $C_{max}$  and TWT.

## 6. EXPERIMENTAL RESULTS

For each of the 20 experimental instances (generated as explained in section 5.1), we use the seven different combinations of desirability weights in Table 2 at both the SSP and MCM levels. Initial pilot run experiments were performed to determine the impact of using desirability at both the SSP and the MCM levels. For the Minifab model, pilot runs suggested that SSP-TWT\_MCM-Des (Global), SSP-TWT-MCM-Des (Simple), and SSP-TWT\_MCM-TWT produced the same results over all 20 instances (i.e., the sequence in which the toolgroups were scheduled in the MSBH was the same for all three approaches). This is not surprising, as the Minifab contains only three toolgroups. Toolgroup 3, which contains only one machine and is subject to sequence-dependent setups (Figure 2) is always determined to be the most critical toolgroup, followed by toolgroup 1 (the batch-processing toolgroup), then toolgroup 2. Thus, the impact of using desirability at the MCM level is not significant for the Minifab model.

However, using desirability at the SSP level produced significantly different results for the Minifab model instances. As the SSP-level results are independent of the approach used at the MCM level, the approaches to be compared reduce to SSP-TWT and SSP-Des. The former represents the first combination in Table 2, while the latter represents the “best” of the remaining six combinations of desirability weights. To determine the best combination, we use the desirability function again, but now externally when the complex job shop has been fully scheduled and all objective functions have been realized.

Let  $TWT_{(best)}$  denote the best (i.e., lowest) value of TWT obtained from using the following three dispatching rules: CR, EDD, and FIFO. As these dispatching rules are expected to perform poorly for TWT as they do not explicitly consider job due dates and/or weights, we use  $TWT_{(best)}$  as our upper bound on TWT. Therefore, let  $G_{C_{max}} = C_{max}_{(min)}$ ,  $G_{TWT} = TWT_{(min)}$ ,  $U_{C_{max}} = C_{max}_{(max)}$ , and  $U_{TWT} = TWT_{(best)}$ . Externally, initially we assume that the decision maker places equal emphasis on both objectives ( $\alpha_{C_{max}} = \alpha_{TWT} = 0.5$ ). Table 3 shows the combined,  $C_{max}$ , and TWT desirabilities for each of the 20 Minifab model instances for three different scheduling approaches (SSP-TWT, SSP-Des (with the combination of weights being chosen via pilot runs as described above), and SSP-Des\* (the schedule

generated using  $\alpha_{C_{max}} = \alpha_{twf} = 0.5$ )) and the three competing dispatching rules (CR, EDD, and FIFO). The bold number in each row is the best desirability value of the corresponding instance.

It is clear that while SSP-TWT is superior in terms of TWT performance, its  $C_{max}$  desirability is very poor when compared to CR and SSP-Des. Therefore, its combined desirability is not good. CR and FIFO have high  $C_{max}$  desirability, but their respective TWT desirabilities over all 20 instances are poor. SSP-Des performs reasonably well for both objectives, and therefore has the best combined desirability for the 20 Minifab problem instances. SSP-TWT shares the best desirability with SSP-Des in only one of the 20 instances, instance 17. A comparison between SSP-Des and SSP-Des\* reveals that SSP-Des has a desirability roughly 9% better than SSP-Des\*. Further, the former performs more consistently than the latter (see standard deviation results). However, SSP-Des is computationally more expensive due to the exhaustive desirability weight search. Therefore, if a quick solution that is “good” in both objectives is desired, SSP-Des\* should be used for Minifab problem instances.

## 7. CONCLUSIONS AND FUTURE RESEARCH

In this paper, we propose a technique for optimizing a single aggregated function involving two objectives (makespan and total weighted tardiness) in a complex job shop. We adapt the MSBH for complex job shops of Mason *et al.* (2002) to include multiple objectives using the desirability function approach of Derringer and Suich (1980). We use the desirability approach at two different levels of the MSBH, the SSP level and the MCM level, and propose five new approaches for scheduling complex job shops. We test our approaches on the Minifab model to the original MSBH (approach “SSP-TWT\_MCM-TWT”) as well as three dispatching rules. Twenty problem instances consisting of 20 jobs each were generated and analyzed. External to the MSBH, we use the desirability function to compare the schedules generated by the different competing approaches. Experimental results show that when equal emphasis is placed on all three objectives, the desirability approach performs significantly better than both the MSBH and dispatching rules. An important conclusion from our experimentation is that the use of a desirability approach at the SSP level consistently produces superior results.

## REFERENCES

- Adams, J., E. Balas, and D. Zawack, “The shifting bottleneck procedure for job shop scheduling”, *Management Science*, 34, 391-401 (1988).
- Cochran, J.K., S.M. Horng, and J.W. Fowler, “A Multi-Population Genetic Algorithm to Solve Multi-Objective Scheduling Problems for Parallel Machines”, *Computers and Operations Research*, 30, , 1087-1102 (2003).
- Dabbas, R., J. Fowler, D. Rollier and D. McCarville, “Multiple response optimization using mixture-designed experiments and desirability functions in semiconductor scheduling”, *International Journal of Production Research*, 41 939-961(2003).
- Derringer, G., and R. Suich, “Simultaneous Optimization of Several Response Variables”, *Journal of Quality Technology*, 12, 214-219(1980).
- El Adl, M.K., A.A. Rodriguez, and K.S. Tsakalis, “Hierarchical modeling and control of re-entrant semiconductor manufacturing facilities”, in *Proceedings of the 35th Conference on Decision and Control*, Kobe, Japan, 1996.
- Esquivel, S.C., S.W. Ferrero, and R.H. Gallard, “Parameter settings and representations in pareto-based optimization for job shop scheduling”, *Cybernetics and Systems*, 33, 559-578(2002).
- Gadkari, A., “Scheduling to reduce total weighted tardiness on parallel machines with ready times”, *Masters Thesis*, Arizona State University (2003).
- Hoogeveen, H., Multicriteria Scheduling, *European Journal of Operational Research*, Article in Press.

- Lima, H., T. Hara, N. Ichimi, and N. Sannomiya, "Autonomous decentralized scheduling algorithm for a job-shop scheduling problem with complicated constraints", in *Proceedings of the 1999 4<sup>th</sup> International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan 366-369 (1999).
- Itoh, K., D. Huang, and T. Enkawa, "Twofold look-ahead search for multi-criterion job shop scheduling", *International Journal of Production Research*, 31(9), 2215-2234 (1993).
- Kacem, I. , S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic", *Mathematics and Computers in Simulation*, 60, 245-276 (2002).
- Law, A.M., and W.D. Kelton, *Simulation Modeling and Analysis*, Mcgraw Hill, Boston, US, 2000.
- Mason, S.J. , J.W. Fowler, and W.M. Carlyle, "A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops", *Journal of Scheduling*, 5(3), 247-262(2002).
- Myers, R., and D. Montgomery, *Response Surface Methodology*, Wiley, New York, 1995.
- Nagar, A. , J. Haddock, and S. Heragu, "Multiple and bicriteria scheduling: a literature survey", *European Journal of Operational Research*, 81, 88-104 (1995).
- Ovacik, I.M. , and R. Uzsoy, *Decomposition methods for complex factory scheduling problems*, Kluwer Academic Publishers, Massachusetts, 1997.
- Pinedo, M.L. , and M. Singer, "A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop", *Naval Research Logistics*, 46, 1-17 (1999).
- Petrovic, S., Fayad, C., "A Fuzzy Shifting Bottleneck Hybridised with Genetic Algorithm for Real-world Job Shop Scheduling", in: C.H.Antunes, and L.C.Dias, (Eds.), *Proceedings of Mini-EURO Conference, Managing Uncertainty in Decision Support Models, MUDSM 2004, Coimbra, Portugal, 22-24 September, 2004*, pp 1-6 (on CD), 2004, SBN: 972-9044-52-X.).
- T'kindt, V.T., and J.-C. Billaut, *Multicriteria scheduling: theory, models and algorithms*, Springer, Berlin, 2002.

## 8. FIGURES AND TABLES

| No. | SSP level    | MCM level                 |
|-----|--------------|---------------------------|
| 1   | Only TWT     | Only TWT                  |
| 2   | Only TWT     | Desirability (Local MCM)  |
| 3   | Only TWT     | Desirability (Global MCM) |
| 4   | Desirability | Only TWT                  |
| 5   | Desirability | Desirability (Simple MCM) |
| 6   | Desirability | Desirability (Global MCM) |

*Table 1.* Different Approaches

| Cmax | TWT  |
|------|------|
| 1    | 0    |
| 0    | 1    |
| 0.5  | 0.5  |
| 0.67 | 0.33 |
| 0.33 | 0.67 |
| 0.83 | 0.17 |
| 0.17 | 0.83 |

*Table 2.* Desirability Weight Settings

| Instance | Combined Desirability |         |          |        |        |        | Makesp n Desirability |         |          |        |        |        | TWT Desirability |         |          |        |        |        |
|----------|-----------------------|---------|----------|--------|--------|--------|-----------------------|---------|----------|--------|--------|--------|------------------|---------|----------|--------|--------|--------|
|          | SSP-TWT               | SSP-Des | SSP-Des* | CR     | EDD    | FIFO   | SSP-TWT               | SSP-Des | SSP-Des* | CR     | EDD    | FIFO   | SSP-TWT          | SSP-Des | SSP-Des* | CR     | EDD    | FIFO   |
| 1        | 0.5995                | 0.8658  | 0.8217   | 0.1038 | 0.0084 | 0.0084 | 0.3697                | 0.7704  | 0.7839   | 1.0000 | 0.7124 | 0.7483 | 0.9721           | 0.9731  | 0.8614   | 0.0108 | 0.0001 | 0.0001 |
| 2        | 0.8643                | 0.9729  | 0.9688   | 0.0916 | 0.0015 | 0.0015 | 0.7471                | 1.0000  | 1.0000   | 0.7541 | 0.0215 | 0.7541 | 1.0000           | 0.9465  | 0.9386   | 0.0111 | 0.0001 | 0.0001 |
| 3        | 0.8621                | 0.8640  | 0.8640   | 0.1105 | 0.0015 | 0.0015 | 0.7432                | 0.7880  | 0.7880   | 1.0000 | 0.0214 | 0.7501 | 1.0000           | 0.9472  | 0.9472   | 0.0122 | 0.0001 | 0.0001 |
| 4        | 0.9062                | 0.9213  | 0.9213   | 0.0100 | 0.0965 | 0.0965 | 0.8486                | 0.8501  | 0.8501   | 0.9967 | 0.8144 | 0.8501 | 0.9678           | 0.9985  | 0.9985   | 0.0001 | 0.0114 | 0.0001 |
| 5        | 0.1343                | 0.8735  | 0.7362   | 0.0100 | 0.0932 | 0.0932 | 0.0180                | 0.9535  | 0.6739   | 0.9934 | 0.8087 | 0.8469 | 1.0000           | 0.8002  | 0.8043   | 0.0001 | 0.0108 | 0.0001 |
| 6        | 0.7056                | 0.9581  | 0.9581   | 0.0099 | 0.0174 | 0.0174 | 0.4979                | 0.9817  | 0.9817   | 0.9817 | 0.0283 | 0.9909 | 1.0000           | 0.9352  | 0.9352   | 0.0001 | 0.0107 | 0.0001 |
| 7        | 0.6149                | 0.9308  | 0.7623   | 0.0087 | 0.0015 | 0.0015 | 0.3781                | 0.9225  | 0.7331   | 0.7625 | 0.0219 | 0.7625 | 1.0000           | 0.9391  | 0.7927   | 0.0001 | 0.0001 | 0.0110 |
| 8        | 0.1718                | 0.9792  | 0.9792   | 0.0096 | 0.0898 | 0.0898 | 0.0307                | 1.0000  | 1.0000   | 1.0000 | 0.9290 | 0.7290 | 0.9614           | 0.9588  | 0.9588   | 0.0001 | 0.0111 | 0.0001 |
| 9        | 0.5360                | 0.9527  | 0.9527   | 0.0092 | 0.0014 | 0.0014 | 0.2974                | 1.0000  | 0.7840   | 0.7463 | 0.0213 | 0.7463 | 0.9662           | 0.9076  | 0.9076   | 0.0001 | 0.0001 | 0.0106 |
| 10       | 0.5672                | 0.8677  | 0.8178   | 0.0086 | 0.0152 | 0.0152 | 0.3217                | 0.7707  | 0.7963   | 0.7463 | 0.0213 | 0.7463 | 1.0000           | 0.9769  | 0.8530   | 0.0001 | 0.0108 | 0.0001 |
| 11       | 0.8845                | 0.9210  | 0.8924   | 0.0089 | 0.0160 | 0.0160 | 0.8008                | 0.9421  | 0.7963   | 0.7963 | 0.0229 | 0.7963 | 0.9770           | 0.9004  | 1.0000   | 0.0001 | 0.0112 | 0.0001 |
| 12       | 0.1510                | 0.8413  | 0.1300   | 0.1069 | 0.0080 | 0.0080 | 0.0228                | 0.8086  | 0.0228   | 1.0000 | 0.6346 | 0.6986 | 1.0000           | 0.8752  | 0.7413   | 0.0114 | 0.0001 | 0.0001 |
| 13       | 0.1376                | 0.9674  | 0.8583   | 0.0094 | 0.0046 | 0.0046 | 0.0189                | 0.9913  | 0.8863   | 0.8863 | 0.2161 | 0.8904 | 1.0000           | 0.9441  | 0.8311   | 0.0001 | 0.0107 | 0.0001 |
| 14       | 0.1335                | 0.9142  | 0.6689   | 0.0094 | 0.0941 | 0.0941 | 0.0178                | 0.9606  | 0.8628   | 0.8772 | 0.8416 | 0.8772 | 1.0000           | 0.8701  | 0.5185   | 0.0001 | 0.0105 | 0.0001 |
| 15       | 0.7947                | 0.8828  | 0.8489   | 0.1128 | 0.0014 | 0.0014 | 0.6315                | 0.9680  | 0.8471   | 0.9965 | 0.0185 | 0.8446 | 1.0000           | 0.8051  | 0.8507   | 0.0128 | 0.0001 | 0.0001 |
| 16       | 0.7171                | 0.8811  | 0.8521   | 0.0100 | 0.0015 | 0.0015 | 0.7501                | 1.0000  | 0.7262   | 0.9948 | 0.0214 | 0.7501 | 0.6855           | 0.7763  | 1.0000   | 0.0001 | 0.0001 | 0.0114 |
| 17       | 0.8451                | 0.8451  | 0.8220   | 0.0090 | 0.0015 | 0.0015 | 0.7945                | 0.7945  | 0.6757   | 0.8021 | 0.0231 | 0.8021 | 0.8989           | 0.8989  | 1.0000   | 0.0001 | 0.0001 | 0.0116 |
| 18       | 0.1516                | 0.9166  | 0.8205   | 0.0083 | 0.0077 | 0.0077 | 0.0230                | 1.0000  | 0.6967   | 0.6822 | 0.5925 | 0.6913 | 1.0000           | 0.8401  | 0.9662   | 0.0001 | 0.0001 | 0.0115 |
| 19       | 0.5274                | 0.8879  | 0.8595   | 0.0087 | 0.0015 | 0.0015 | 0.2781                | 1.0000  | 0.7561   | 0.7633 | 0.0220 | 0.7633 | 1.0000           | 0.7883  | 0.9771   | 0.0001 | 0.0001 | 0.0108 |
| 20       | 0.6139                | 0.9431  | 0.8503   | 0.0087 | 0.0015 | 0.0015 | 0.3769                | 0.9520  | 0.7501   | 0.7501 | 0.0214 | 0.7501 | 1.0000           | 0.9342  | 0.9638   | 0.0001 | 0.0001 | 0.0102 |
| Average  | 0.5459                | 0.9093  | 0.8192   | 0.0332 | 0.0232 | 0.0232 | 0.3983                | 0.9227  | 0.7807   | 0.8778 | 0.2806 | 0.8049 | 0.9715           | 0.9008  | 0.8923   | 0.0030 | 0.0039 | 0.0045 |
| Std.Dev  | 0.2923                | 0.0436  | 0.1802   | 0.0428 | 0.0364 | 0.0364 | 0.3106                | 0.0885  | 0.2081   | 0.1125 | 0.3477 | 0.0790 | 0.0717           | 0.0675  | 0.1178   | 0.0052 | 0.0053 | 0.0055 |

Table 3. Minitab model: Desirability values for the approaches with weights  $\alpha_{c\max} = \alpha_{tw} = 0.5$  for the combined desirability

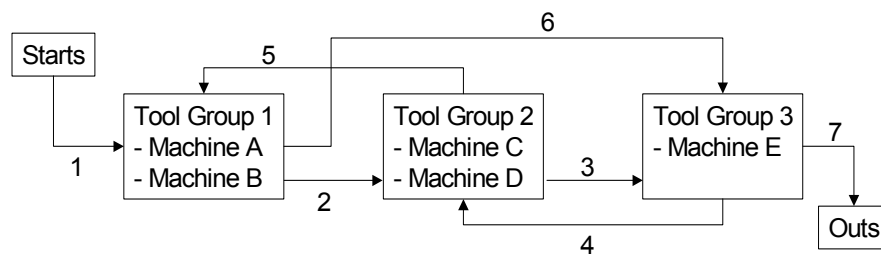


Figure 1. The Minifab Model (El Adl *et al.* 1996, Mason *et al.* 2002)



# MINIMIZING MAKESPAN WITH MULTIPLE ORDERS PER JOB IN A TWO MACHINE FLOWSHOP WITH ITEM PROCESSING

Jeffrey D. Laub, John W. Fowler, and Ahmet B. Keha  
*Arizona State University*

**Abstract:** New semiconductor wafer fabrication facilities use Front Opening Unified Pods (FOUPs) as a common unit of wafer transfer. Since the number of pods is limited due to high costs, and the increase in wafer size to 300-mm reduces the number of wafers required for a customer order, combining multiple orders into a single job is a necessity for efficient production. We investigate the multiple orders per job scheduling problem in the context of a 2-machine flowshop. A lower bound for the makespan performance measure is provided. An optimization model is presented that addresses both job formation and job sequencing.

**Key words:** flowshop, scheduling, multiple orders per job, optimization, semiconductor manufacturing, makespan, lot streaming

## 1. INTRODUCTION

The semiconductor manufacturing industry is one of the largest industries in the world, with annual revenues exceeding US \$200 Billion. With the extraordinary size of this market comes stiff competition and the need for manufacturers to continually improve their processes and equipment. One recent response to this challenge has been the increase of silicon wafer size from 200-mm to 300-mm. This results in approximately a 125% increase per wafer. However, since production lots typically contain 25 wafers, the lots are heavier than what can be physically carried safely between wafer processing tools. To address this, the industry has established a standard for the Front-Opening Unified Pod (FOUP) to transfer wafer lots between tools in a 300-mm wafer fab. These pods provide a clean atmosphere to prevent contamination of the wafer surface.

The increased wafer surface area results in a reduction in the number of wafers required to fill a customer's order. Assigning one FOUP to each order is a costly under-utilization of this scarce resource. Also, the processing time required by some operations is independent of the number of wafers. Sparsely filling FOUPs with individual wafers would adversely affect overall production efficiency. Semiconductor manufacturers using 300-mm wafers will need to group orders from different customers into a single FOUP, or job, which can then be routed to the appropriate processing tool. Mason, Qu, Kutanoglu, and Fowler [1] and Kutanoglu, Tanrisever, and Mason [2] and have addressed this problem domain, and refer to it as "multiple orders per job" (*moj*) scheduling problems.

Multiple orders per job problems are complex due to the interaction of two decisions that must be made, namely, job formation from orders and job scheduling. Further difficulties are imposed by the

restricted size of each job as well as the limited number of jobs (pods) available. Finally, there are at least two types of machines to consider for job processing. In *single lot processing*, such as wet sink processing, the processing time is independent of the number of items in the job. In *single item processing*, the processing time of a job is a function of the time required per item (wafer) as well as the number of items in the job. The effects of these machine properties is captured in the constraint field ( $\beta$ ) of scheduling notation as  $moj(lot)$  and  $moj(item)$  respectively. Note that the additional option of batching the formed jobs would be represented by  $moj(\bullet),batch$ .

At this time, publications addressing the multiple orders per job problem deal with single machine configurations and either Total Completion Time or Total Weighted Completion Time performance measures. As semiconductor manufacturing processes typically entail a series of processes, the natural extension of analysis is the consideration of flowshops. This observation, plus the fact that the makespan is a typical performance measure of interest, motivates the proposed investigation of  $Fm | moj(\bullet) | Cmax$  and related problems. The results of this paper establish a lower bound on the makespan for multiple orders per job problems in a 2-machine flowshop.

## 2. PROBLEM DEFINITION

We assume that a set of orders  $O=\{o_k\}$ ,  $k=1..H$  are available at time zero for processing and scheduling on  $m$  sequential machines. Each order has a size  $s_k$ , representing the number of items (wafers) in the order. In general, the processing time of a single item is dependent on its order and the machine performing the processing. In our problem, we restrict the item processing time to be independent of the order. Without loss of generality, each item requires one unit of processing time on the slowest machine. The processing time of each item is the same on any given machine, but the machine speeds may be different.

Each order must be assigned to exactly one of  $F$  jobs. Each job has a size capacity of  $K$ , and multiple orders can be assigned to a single job as long as the sum of order sizes does not exceed  $K$ , and no order is divided among multiple jobs (In the case where the size of an order exceeds  $K$ , we assume the order is first divided into separate orders, each of which is of size  $K$ , except for possibly one). No preemption of jobs or orders is allowed.

As discussed earlier, the processing time of a job depends on the type of machine. In an item-processing machine, denoted  $moj(item)$ , the job processing time is the sum of the orders' individual item processing times for that machine. The task is to assign the orders to jobs and schedule the jobs to minimize the makespan of the entire set of orders through a 2-machine flowshop.

To accommodate the problem description above, we use the following notation.

- $m$  : number of machines
- $M_i$  : machine  $i$ ,  $i \in \{1..m\}$
- $K$  : job capacity
- $F$  : maximum number of jobs allowed (FOUPs)
- $H$  : number of orders
- $o_k$  : order  $k$ ,  $k \in \{1..H\}$
- $\rho_{ik}$  : processing time for order  $k$  on machine  $i$ .
- $\tau_{ik}$  : processing time per item in order  $k$  on machine  $i$ .
- $s_k$  : size of order  $k$ . We assume  $s_k \leq K$ .
- $p_{i,j}$  : processing time for job  $j$  on machine  $i$ .  
 $= \sum \rho_{i,k}$  for  $o_k \in \text{Job } j$  for  $moj(item)$  machines

### 3. LITERATURE REVIEW

As the *moj* problem set is relatively new, there are few papers published on this subject. Mason, *et al.* [1] authored one of the most recent articles, and establishes much of the background motivation for the problem domain. Their paper provides a linear integer program for the single machine *moj*(•) Total Weighted Completion Time problems. Due to the lengthy execution time of this model, they propose and analyze combinations of 5 order sequencing rules, 2 job filling directions to select the orders from the sequence, and 6 order-to-job assignment rules specifying the job construction stage. Their results identify which heuristic combinations provide the best performance measurement while adhering to short execution times, dependent on problem instance characteristics.

Another recent paper by Kutanoglu, *et al.* [2] addresses the single machine *moj*(item) Total Completion Time problem. For the uncapacitated version, they devise an improvement-based algorithm and a lot-size-based algorithm for solving the problem to optimality. As the complexity of the former is exponential in the number of orders, it becomes ineffective at approximately 35 jobs. The latter is polynomial in the number of orders. The capacitated version is sufficiently complex to abandon attempts to achieve an optimal algorithm, and the authors provide a heuristic based on the prior optimal solutions by removing the capacity infeasibilities while preserving as much of the original schedule as possible. Their experiments demonstrate the effects of various problem parameters on the performance of the heuristic.

Prior to these contributions, another paper addressing a similar problem was Dobson and Nambimadom's [3] article on batch loading and scheduling. They address the mean weighted flow time objective for a single machine. However, the batch processing times depend only on the family, not the number or volume of jobs in a batch, and there is no restriction on the number of batches constructed.

We will demonstrate that by removing several of the restrictions of the multiple orders per job problem, its relaxation resembles a form of lot streaming. See Chang and Chiu [4] for a comprehensive survey of lot streaming results.

To the authors' knowledge, this is the first paper to address the multiple orders per job problem domain in the context of flowshops.

### 4. ITEM PROCESSING

For item-processing, we demonstrate that we only need to consider solutions that use all the available jobs ( $F$ ). This result is not restricted to the 2-machine case.

**Theorem 1.** For the  $F_m \mid \text{moj}(\text{item}), \text{prmu} \mid C_{\max}$  problem, there is an optimal schedule that uses  $\min(F, H)$  jobs.

**Proof.** Suppose there is no optimal schedule that uses  $\min(F, H)$  jobs. Then there is an optimal schedule  $S$  in which at least one job  $j$  contains more than one order, and there are  $n$  jobs where  $n < \min(F, H)$  (see Figure 1). Construct  $S'$  from  $S$  by removing one order from this job (creating job  $j'$ ) and creating a new job  $j''$  containing just the one order. Since  $p_{ij} = p_{ij'} + p_{ij''}$ , the original job  $j$  in  $S$  can be replaced by jobs  $j'$  and  $k$  in  $S'$  without increasing  $C_{\max}(S')$  (see Figure 2). This can be repeated until  $n$  is at least  $\min(F, H)$ . Thus the original assumption is invalid.

<<Figure 1>>

<<Figure 2>>

Since the case for  $H \leq F$  is solved by placing each order in a separate job, we will assume that  $H > F$ . We also assume that the processing time of all items is the same on a given machine ( $\tau_{ik} = \tau_i$ ). Consider the following instance:

$\tau_{1,k} = 1$       All items require 1 time unit on machine 1  
 $\tau_{2,k} = 2$       All items require 2 time units on machine 2  
 $s_1 = 3$         The size of order 1 is 3 items  
 $s_2 = 4$         The size of order 2 is 4 items  
 $s_3 = 5$         The size of order 3 is 5 items  
 $J_1 = \{o_1, o_2\}$     Job 1 consists of orders 1 and 2  
 $J_2 = \{o_3\}$         Job 2 consists of order 3  
 Then the job processing times on each machine are as follows:

<<Table 1>>

Essentially, the machines have different rates, but the same machine rate applies to all items. This will result in jobs whose processing times equal the number of items in the job times the relative machine factor. Therefore, once the job composition has been determined, the sequencing problem becomes  $F2 \mid \text{prmu}, p_{i,j} = p_j/v_i \mid C_{\max}$ , where  $v_i$  is the relative machine speed. Applying this notation to the example above, we have  $p_1 = 7$ ,  $p_2 = 5$ ,  $v_1 = 1$ ,  $v_2 = 1/2$ . This leads to the following result for the uncapacitated problem.

**Lemma 1.** The  $F2 \mid \text{moj}(\text{item}), \text{prmu}, \tau_{ik} = \tau_i \mid C_{\max}$  problem (with  $K = \infty$ ) is equivalent to finding the best assignment of orders to jobs for which the  $F2 \mid \text{prmu}, p_{i,j} = p_j/v_i \mid C_{\max}$  problem solution has the best  $C_{\max}$ .

## 5. OPTIMAL UNCAPACITATED JOB SIZES

The flexibility in these moj problems is the ability to somewhat control the job processing times via the chosen job composition (order assignment). For the 2-machine case, since we know that the SPT-LPT algorithm of Johnson [5] is optimal, the goal is to select the job composition for which the optimal solution is better than the optimal solutions for all other job compositions. Once job composition is determined, then optimal job sequencing is easily determined. In general, the task is to characterize the properties of a job mix that best exploits the optimal solutions to the problems. To find the optimal job sizes, we first consider the problem as if orders were infinitely divisible, and the total order processing time could be distributed arbitrarily among the jobs.

Since there is no idle time for  $M_1$ , and the sequence of job completion is irrelevant, the goal is to minimize the idle time on  $M_2$ . Intuitively, we'd like to get the first job finished on  $M_1$  quickly so that  $M_2$  can start work, and have no other idle time for  $M_2$ . Assuming in the first case that  $M_2$  is slower than  $M_1$ , we let the  $M_1$  speed,  $v_1$ , equal 1 and let  $c = 1/v_2$ , so that  $c > 1$ . Once job 1 has completed processing on  $M_1$ , since we know it will take  $cp_{1,1}$  time on  $M_2$ , the best choice for  $p_{1,2}$  is  $cp_{1,1}$ . In general, we would like  $p_{1,k+1} \leq p_{2,k}$ . This results in no idle time on  $M_2$  other than the wait for job 1 to finish on  $M_1$ . So we would then like to have as small a value for  $p_{1,1}$  as possible. This is depicted in the following diagram.

<<Figure 3>>

If we let  $P_1$  equal the total processing time requirement for all jobs on  $M_1$ , then the recursive relationship results in the following.

$$p_{1,k} = c^{k-1} p_{1,1} \text{ for } 2 \leq k \leq n$$

$$\sum_{j=1}^n p_{1,j} = P_1 = p_{1,1} \sum_{j=0}^{n-1} c^j$$

Thus, the optimal selection of processing times is geometric. The result, analogous to the result of Potts and Baker [6] regarding lot streaming, is as follows.

**Lemma 2:** For a given number of jobs  $n$ , and a given total  $P_1$  of processing times of the  $n$  jobs on machine 1, and a relative machine 2 rate of  $v_2 < 1$ , the geometric distribution (using factor  $c=1/v_2$ ) of this total produces the best  $C_{max}$  possible for  $F2 | \text{prmu}, p_{i,j} = p_j/v_i | C_{max}$ , among all partitions of  $P_1$  over  $n$ , i.e.,

$$p_{1,j} = P_1 \frac{c^{j-1}(c-1)}{c^n - 1} \quad (1)$$

**Lemma 3:** The optimal  $C_{max}$  is

$$P_1 \left( \frac{c^{n+1} - 1}{c^n - 1} \right) \quad (2)$$

A similar result holds when  $v_2 > 1$ , with jobs sequenced by decreasing processing time.

## 6. OPTIMAL CAPACITATED JOB SIZES

To apply this result to the *moj(item)* problem, suppose we have the sum of order processing times = 15 on machine  $M_1$ , and  $M_2$  runs 2 times as slow as  $M_1$ , and the maximum number of jobs (FOUPs) we can construct is 4. Then we want to allocate the orders to jobs (if possible) so that

$$p_{1,j} = P_1 \frac{c^{j-1}(c-1)}{c^n - 1} = (15) \frac{2^{j-1}}{15} = 2^{j-1}$$

The partitioning of  $P_1$  that meets this requirement is (1, 2, 4, 8), with  $C_{max} = 31$ . However, this does not consider the possible restriction imposed by the limiting job capacity,  $K$ . If  $K = 6$ , then the optimum partition is infeasible. If job 4 is reduced to the maximum processing time, the remaining jobs become a subproblem, with  $n'=3$ , and  $P' = 9$ . This produces a distribution of (1.29, 2.57, 5.14, 6.00) for the complete problem, with a  $C_{max}$  of 31.29.

Based on this strategy, the following theorem determines the optimal job sizes  $s$  for a given flowshop with machine 2 time factor equal to  $c$ , a total number of items  $P_1$ , a maximum number of jobs  $n$ , and a job capacity limit  $K$ .

**Theorem 2:** For  $F2|moj(item)|C_{max}$  problem, the following equations determine the optimal job processing times  $p_{i,j}$  for a given flowshop with machine 2 time factor  $c>1$ , a total machine 1 processing time  $P_1$ , a maximum number of jobs  $n$ , and a job capacity limit  $K$ , where job size is equal to the job processing time.

If

$$b = \max \left\{ x \mid (P_1 - xK) \left( \frac{c^{n-1-x}(c-1)}{c^{n-x} - 1} \right) > K \right\}$$

for integer(x), ( $b$  is the number of capacitated jobs) then the optimal job processing times satisfy:

$$p_{1,j} = (P_1 - bK) \frac{c^{j-1}(c-1)}{c^{n-b} - 1} \quad 1 \leq j \leq n-b \quad (3)$$

$$p_{1,j} = K \quad n-b < j \leq n \quad (4)$$

Jobs with these processing times result in the following Cmax.

$$C_{\max} = (P_1 - bK) \left( \frac{c^{n-b+1} - 1}{c^{n-b} - 1} \right) + cbK \quad (5)$$

The structure of the proof is as follows. First, we show that for the processing times in equations (3) and (4), equation (5) specifies the corresponding value of Cmax. Next, we show the case for  $b=1$ , i.e., that the equations hold when only one job needs to be reduced in size from the geometric optimum to size  $K$ . We then use this fact as the basis of induction on  $b$ . We then show that the  $p_{i,j}$  defined by equations (3) and (4) sum to  $P$ .

Step 1. We show that equation (5) holds.

The Cmax for the job sequence defined by equations (3) and (4) is equal to the Cmax for the uncapacitated jobs plus the total processing time for the remaining jobs on machine 2, which takes  $cbK$  time. From Lemma 3, the Cmax for the uncapacitated jobs is

$$(P_1 - bK) \left( \frac{c^{n-b+1} - 1}{c^{n-b} - 1} \right)$$

Step 2. We prove that if  $P_1$  is large enough to require capacitation of exactly one job, then the hypothesis of Theorem 2 is true. This is captured by the following lemma.

Lemma 4: Let

$$(P_1 - K) \left( \frac{c^{n-2}(c-1)}{c^{n-1} - 1} \right) > K$$

$$(P_1 - 2K) \left( \frac{c^{n-3}(c-1)}{c^{n-2} - 1} \right) \leq K$$

Then equations (3) and (4) define the optimal processing times for the Cmax objective.

Proof (by contradiction): Suppose this is not the case. Then there is a superior set of  $p_{1,j}$  where either (1)  $p_{1,n} < K$ , or (2) the  $p_{1,j < n}$  are not geometric per equation (3).

Case 1: Suppose  $p_{1,n} < K$ .

Let  $p_{1,n} = K - \varepsilon$ . Let  $C_{\max}'$  be the optimal makespan for the first  $n-1$  jobs. From Lemma 3, we have

$$C_{\max}' = (P_1 - p_{1,n}) \left( \frac{c^n - 1}{c^{n-1} - 1} \right)$$

So the makespan for the entire job set is

$$C_{\max} = (P_1 - p_{1,n}) \left( \frac{c^n - 1}{c^{n-1} - 1} \right) + cp_{1,n}$$

Thus,

$$C_{\max} = (P_1 - K) \left( \frac{c^n - 1}{c^{n-1} - 1} \right) + cK + \left( \frac{c^n - 1}{c^{n-1} - 1} - c \right) \varepsilon$$

But this is larger than the  $C_{\max}$  of equation (5).

Case 2: Suppose that the first  $n-1$  job processing times are not geometric per equation (3).

The processing time distributed among these jobs is  $P_1 - K$ . From Lemma 2, the optimal distribution is

$$p_{1,j} = (P_1 - K) \frac{c^{j-1}(c-1)}{c^{n-1} - 1}$$

This is exactly equation (3) for  $b=1$ , thus contradictory.

End of Lemma 4 Proof

Step 3: We have shown that the theorem is true for the  $b=1$  case. We now complete the proof by induction on  $b$ . We will demonstrate that if the theorem holds for any  $b < q$ , then it holds for  $b = q$ .

Suppose  $b = q$ . Then

$$(P_1 - qK) \frac{c^{n-q-1}(c-1)}{c^{n-q} - 1} > K \quad (6)$$

$$(P_1 - (q+1)K) \frac{c^{n-q-2}(c-1)}{c^{n-q-1} - 1} \leq K \quad (7)$$

We will now show that the optimal distribution must have at least  $q$  jobs of processing time  $K$ . Assume there are only  $q-1$  such jobs. Then consider the remaining  $n' = n - q + 1$  jobs. The total processing time for this set is  $P' = P_1 - (q-1)K$ . We show that for this set of jobs,  $b' = 1$ . From equations (6) and (7) we have

$$(P' - K) \frac{c^{n'-2}(c-1)}{c^{n'-1} - 1} = (P_1 - qK) \frac{c^{n-q-1}(c-1)}{c^{n-q} - 1} > K$$

$$(P'-2K) \frac{c^{n'-3}(c-1)}{c^{n'-2}-1} = (P_1 - (q+1)K) \frac{c^{n-q-2}(c-1)}{c^{n-q-1}-1} \leq K$$

Therefore,  $b'=1$ .

This satisfies the induction hypothesis for  $b = q - 1$ . So the optimal distribution for the reduced job set is defined by equations (3) and (4) for  $P'$ ,  $m'$ , and  $n'$ . Looking at the last job in this sequence, we have

$$p_{1,n'} = p_{1,n-(q-1)} = (P_1 - qK) \frac{c^{n-q}(c-1)}{c^{n-q+1}-1} > (P_1 - qK) \frac{c^{n-q-1}(c-1)}{c^{n-q}-1} > K$$

This violates the assumption that there were less than  $q$  jobs of size  $K$  or greater. Thus, there must be at least  $q$  jobs of size  $K$ .

To show that there are no more than  $q$  jobs of size  $K$ , we suppose that there are  $q+1$  jobs of size  $K$  and demonstrate a contradiction. Consider the job set obtained by removing  $q$  jobs of processing time  $K$ . The last job in this remaining sequence has processing time  $K$ , and the total processing time is  $P'=P-qK$ . By Lemma 2, the last job should have processing time

$$p_{1,n-q} = (P_1 - qK) \frac{c^{n-q-1}(c-1)}{c^{n-q}-1}$$

But by equation (6), this is larger than  $K$ , which is a contradiction. Therefore, the original job set must have exactly  $q$  jobs with size  $K$ .

With  $q$  jobs of size  $K$ , the remaining processing time is  $P'=P-qK$ . Applying Lemma 2, equation (3) is met for  $b=q$ , and the induction is completed.

Step 4. We now show that  $\sum_{j=1}^n p_{1,j} = P_1$

$$\sum_{j=1}^n p_{1,j} = \sum_{j=1}^{n-b-1} p_{1,j} + bK = (P_1 - bK) \frac{c-1}{c^{n-b}-1} \sum_{j=0}^{n-b-1} c^j + bK = (P_1 - bK) \frac{c-1}{c^{n-b}-1} \frac{c^{n-b}-1}{c-1} + bK = P_1$$

*End of Theorem.*

A similar result holds when  $c < 1$ .

## 7. MATHEMATICAL FORMULATION

The following Mixed Integer Programming model represents the  $F_m \mid \text{moj}(\text{item}), \text{prmu}, \tau_{ik} = \tau_i \mid C_{\max}$  problem.

Decision Variables:

$x_{i,j}$  = start time of job  $j$  on machine  $i$ , for  $i \in \{1..m\}, j \in \{1..F\}$

$y_{j,j'}$  = 1 if job  $j$  runs before job  $j'$ , 0 otherwise, for  $j \in \{1..F\}, j' \in \{j+1..F\}$

$z_{k,j}$  = 1 if order  $k$  is assigned to job  $j$ , 0 otherwise, for  $k \in \{1..H\}, j \in \{1..F\}$ .



$p_{i,j}$  = processing time of job  $j$  on machine  $i$ .

Constants:

$m$  = number of machines

$s_k$  = number of items in order  $k$ , for  $k \in \{1..H\}$ .

$\tau_i$  = processing time for a single item on machine  $i$ , for  $i \in \{1..m\}$ ,  $\tau_1=1$ .

$K$  = maximum size of each job, i.e., maximum sum of the sizes of orders within a job.

$F$  = maximum number of jobs allowed

$M$  = very large number, e.g.,  $KF \sum_{i=1}^m \tau_i$ .

Objective Function: Minimize  $C_{max}$

Constraints:

[1] The processing time of a job is equal to the size of the job (sum of order sizes) times the item processing time.

$$p_{i,j} = \tau_i \sum_{k=1}^H s_k z_{k,j} \text{ for } i \in \{1..m\}, j \in \{1..F\}$$

[2] The objective function is the maximum of all job completion times.

$$C_{max} \geq x_{m,j} + p_{m,j} \quad \text{for } j \in \{1..F\}$$

[3] No two jobs  $j$  and  $j'$  can be scheduled on the same machine at the same time

$$\begin{aligned} x_{i,j} + p_{i,j} &\leq x_{i,j'} + M(1 - y_{j,j'}) && \text{for } i \in \{1..m\}, j \in \{1..F\}, j' \in \{j+1..F\} \\ x_{i,j'} + p_{i,j'} &\leq x_{i,j} + M y_{j,j'} && \text{for } i \in \{1..m\}, j \in \{1..F\}, j' \in \{j+1..F\} \end{aligned}$$

[4] A job must complete on machine  $i$  before it begins on machine  $i+1$ .

$$x_{i,j} + p_{i,j} \leq x_{i+1,j} \quad \text{for } i \in \{1..m-1\}, j \in \{1..F\}$$

[5] Each order is assigned to exactly one job.

$$\sum_{j=1}^F z_{k,j} = 1 \text{ for } k \in \{1..H\}$$

[6] No job size exceeds  $K$ .

$$\sum_{k=1}^H s_k z_{k,j} \leq K \text{ for } j \in \{1..F\}$$

Sample problem instances were prepared using the parameters of the experiment design described in Mason, *et al.* [1]. Using the 15-order case ( $H=15$ ), the levels for the number of jobs  $F$ , the job capacity  $K$ , and the distribution of order sizes are determined by the following table, where  $v \in \{3,5\}$  and  $B \in \{1,2\}$ .

<<Table 2>>

Ten instances for each of the four combinations of  $v$  and  $B$  were generated, with the following results. The optimal makespan is compared to the lower bound from Theorem 2. The execution times are in seconds on a 2.4GHz personal computer.

<<Table 3>>

A small increase in the number of available jobs ( $F$ ) from 7 to 8 in Test Set #2 resulted in execution times that were approximately 30 to 60 times longer. In some cases, the runs were terminated after hours of processing without making progress on the best solution found.

## 8. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced the flowshop configuration of the multiple orders per job problem domain. This new problem domain represents the challenges faced by semiconductor manufacturing plants where a fixed quantity of expensive transport equipment must be used effectively to keep production costs low and deliveries on time. The problem requires aggregating orders into jobs, and sequencing those jobs in a flowshop under the restrictions of a limited job capacity, a fixed number of jobs, and non-identical order sizes. Equations are provided for the optimal mix of job sizes to establish a lower bound for the makespan performance criterion. A mixed integer programming model is defined that accounts for the optimal assignment of orders to jobs as well as job sequencing.

The properties of the multiple orders per job problem, especially in a flowshop configuration, provides for a rich problem set. The compute time for calculating optimal solutions for these complex problems is excessive even for relatively small problem instances. The authors are currently investigating using the lower bound in a branch and bound algorithm, and are developing heuristics for various configurations of flowshops and performance measures for the multiple orders per job problem set. We expect that these heuristics will leverage prior work in single-machine counterparts, possibly in conjunction with metaheuristics such as genetic algorithms and Tabu search.

## 9. BIBLIOGRAPHY

- [1] S. J. Mason, P. Qu, E. Kutanoglu, and J. W. Fowler, "The single machine multiple orders per job scheduling problem", ASU Working Paper ASUIE-ORPS-2004-004, 2004.
- [2] F. Tanrisever, E. Kutanoglu, and S.J. Mason, "Forming and Scheduling Jobs from Multiple Orders with Different Attributes: A Computational Study of Special Cases," Proceedings of 13th Industrial Engineering Research Conference, IIE, Houston, TX, May, 2004.
- [3] G. Dobson and R.S. Nambimadom, The batch loading and scheduling problem, *Operations Research*, 2001, Vol. 49, No. 1, 52-65.
- [4] Jen Huei Chang and Huan Neng Chiu, A comprehensive review of lot streaming, *International Journal of Production Research*, 2005, Vol. 43, No. 8, 1515-1536.
- [5] S. M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly*, 1954, Vol. 1, pp 61-68.
- [6] C. N. Potts and K. R. Baker, Flow Shop Scheduling with Lot Streaming, *Operations Research Letters*, 1989, Vol. 8, pp 297-303.

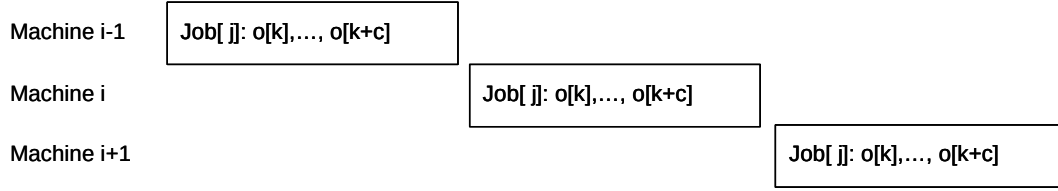


Figure 1. Schedule S with n jobs,  $n < \min(F,H)$

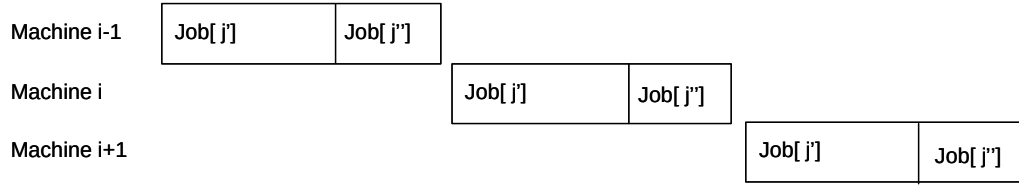


Figure 2. Schedule S' with n+1 jobs

Table 1. Job Processing Times

|    | J1 | J2 |
|----|----|----|
| M1 | 7  | 5  |
| M2 | 14 | 10 |

|       |                  |                    |                    |                    |
|-------|------------------|--------------------|--------------------|--------------------|
| $M_1$ | $J_1: p_{1,1}$   | $J_2: c p_{1,1}$   | $J_3: c^2 p_{1,1}$ | $J_4: c^3 p_{1,1}$ |
| $M_2$ | $J_1: c p_{1,1}$ | $J_2: c^2 p_{1,1}$ | $J_3: c^3 p_{1,1}$ | $J_4: c^4 p_{1,1}$ |

Figure 3. Optimal Job Sizes

Table 2. Experiment Parameters

|              |                            |
|--------------|----------------------------|
| Order Sizes  | $DU[v-(v+1)/2, v+(v+1)/2]$ |
| Capacity (K) | $12B+1$                    |
| Jobs (F)     | $\text{ceiling}(Hv/12B)+1$ |

Table 3. Experiment Results

| Test Set | K  | F | H  | Order Range | Avg Cmax/LB | Avg Time |
|----------|----|---|----|-------------|-------------|----------|
| 1        | 13 | 5 | 15 | 1..5        | 1.018       | <1       |
| 2        | 13 | 7 | 15 | 2..8        | 1.021       | 141      |
| 3        | 25 | 3 | 15 | 1..5        | 1.004       | <1       |
| 4        | 25 | 4 | 15 | 2..8        | 1.019       | <1       |

# A Sequential Pair Exchange Approach for Solving Single-Machine Scheduling Problems with General Earliness-Tardiness Penalties

Shih-Wei Lin and Shuo-Yan Chou

<sup>1</sup>Department of Information Management Hua-Fan University, Taipei, Taiwan,

<sup>2</sup>Department of Industrial Management, Taiwan University of Science and Technology, Taipei, Taiwan

**Abstract:** This study focuses on a class of single-machine scheduling problems with a common due date where the objective is to minimize the total earliness-tardiness penalty for the jobs. A sequential pair exchange approach utilizing a job exchange procedure and three previously established properties in common due-date scheduling was developed and tested with a set of benchmark problems. The developed approach generates results better than not only those of the existing dedicated heuristics but also in many cases those of meta-heuristic approaches. And the developed approach performs consistently well in various job settings with respect to the number of jobs, processing time and earliness-tardiness penalties for the jobs.

**Key words:** Common due date; earliness-tardiness penalties; single-machine scheduling; heuristics.

## 1. INTRODUCTION

The just-in-time (JIT) philosophy involves producing goods only when necessary. Owing to the wide adoption of this philosophy in recent decades, scheduling problems for meeting the due date requirements have been investigated extensively, including those with general earliness-tardiness penalties about a common due date. For such problems, the due date and the penalties are negotiated in advance, when a customer purchases a bundle of goods. Many practical and industrial applications have such requirements. Another application can be found in manufacturing systems where all parts must be ready by a pre-determined date for final assembly and with pre-determined penalty functions.

When scheduling on a single machine about a common due date, at most one job can be completed exactly on the due date. Other jobs have to be completed either earlier or later than the due date. To meet the just-in-time requirement, jobs completed before the common due date incur earliness penalties, while jobs completed after the common due date incur tardiness penalties. The earliness penalties include holding costs for finished goods, deterioration of perishable goods and opportunity cost. The tardiness penalties are the back logging cost which includes performance penalties, loss of sales and loss of goodwill.

The common due date scheduling problem with the earliness-tardiness penalties has been proved to be NP-complete [3]. Therefore, most of such problems were solved by using techniques, including heuristics, branch-and-bound procedures, dynamic programming and meta-heuristics. This study utilizes a sequential pair exchange procedure to establish a heuristic, which results in better solutions than those from the previously established approaches.

The rest of this paper is structured as follows. The next section presents the problem formulation and a short review of literature on single-machine scheduling about a common due date. Section 3 elaborates the proposed approach for solving the scheduling problem with general earliness-tardiness penalties about a common due date. Section 4 compares experimental results with those from existing approaches. Conclusions are drawn in the last section.

## 2. PROBLEM FORMULATION AND LITERATURE REVIEW

The single-machine scheduling problem with general earliness-tardiness penalties about a common due date is defined as follows. There are  $n$  jobs each of which requires exactly one operation are scheduled on a single machine about the common due date  $d$ . For each job  $i$ , the processing time  $p_i$ , the penalty per unit time of earliness  $\alpha_i$ , and the penalty per unit time of tardiness  $\beta_i$  are given. The penalty functions can be classified as linear versus nonlinear, uniform versus varying, and penalty differences among jobs versus penalty differences between earliness and tardiness. Problems with different types of objective functions are reviewed in [1]. In this paper, the earliness and the tardiness penalties for a job are assumed to be linear with respect to the length of the completion time apart from the common due date. Other assumptions include no pre-emption of jobs, all jobs being available for processing at time zero, and the machine being available for work at all times.

A penalty of  $\alpha_i E_i$  incurs when job  $i$  is completed  $E_i$  time units earlier than the common due date; whereas a penalty of  $\beta_i T_i$  incurs when it is completed  $T_i$  time units later than the common due date. Let  $C_i$  be the completion time of job  $i$ . The earliness  $E_i$  and tardiness  $T_i$  can be calculated by  $\max\{d - C_i, 0\}$  and  $\max\{C_i - d, 0\}$ , respectively, for  $i=1, \dots, n$ . The objective is to minimize the total earliness-tardiness penalty, which is defined as the sum of the penalty costs for all the jobs, that is,  $f(S) = \sum_{i=1}^n (\alpha_i E_i + \beta_i T_i)$ .

The common due date problems can be divided into two classes: unrestricted and restricted. In the unrestricted case, the common due date is a decision variable or has no influence on the optimal sequence even if the common due date is known. On the other hand, in the restrictive case, the common due date is fixed and may influence the optimal job sequence.

Due to the nature of the complexity in the problem, global optimal solutions cannot be easily obtained when the problem size is large. Researchers apply polynomially or pseudo-polynomially solvable algorithms to solve for special cases. A comprehensive survey on the common due date assignment and scheduling problems can be found in [4].

For the unrestrictive problem, Cheng and Gupta [3] conducted a survey about models in which due dates are decision variables. Problem with the common due date  $d$  being a decision variable and  $\alpha_i = \beta_i = 1$ , Kanet [11] constructed a polynomially solvable algorithm by a matching algorithm whose time complexity is of  $O(n \log n)$ . Panwalkar et al. [15] proposed a polynomial bound scheduling algorithm with time complexity of  $O(n \log n)$  for solving  $\alpha_i = \alpha$  and  $\beta_i = \beta$  where  $\alpha \neq \beta$  for  $i=1, \dots, n$ . Hall and Posner [8] developed an  $O(n)$  algorithm for solving the problem with  $\alpha_i = \beta_i = p_i$  for  $i=1, \dots, n$ , and an  $O(n \sum p_i)$  dynamic algorithm for solving the case where  $\alpha_i = \beta_i$  for  $i=1, \dots, n$ . Lee et al. [13] presented a dynamic algorithm with time complexity of  $O(n \sum p_i)$  for the special case where  $p_i / \alpha_i < p_j / \alpha_j$  implies  $p_i / \beta_i < p_j / \beta_j$  for all  $i$  and  $j$ .

Hall et al. [7] proved that the restrictive common due-date problem is NP-hard even if  $\alpha_i = \beta_i$  for  $i=1, \dots, n$ , and proposed an  $O(n^2 \sum p_i)$  pseudo-polynomial algorithm to solve this case. Hoogeveen and van de Velde [10] also proposed a pseudo-polynomial dynamic optimization algorithm whose running time is  $O(n^2 \sum p_i)$ . Kahlbacher [12] extended the results of Hall et al. [7], and constructed an  $O(n \sum p_i)$  pseudo-polynomial algorithm to the restrictive problem with  $\alpha_i = \alpha$  and  $\beta_i = \beta$  for  $i=1, \dots, n$ . Szwarc [17] developed a new branch and bound algorithm for problems with  $\alpha_i = \alpha$  and  $\beta_i = \beta$  for  $i=1, \dots, n$ . Lee et al. [13] presented a dynamic algorithm with time complexity of  $O(n \sum p_i + n^2 d)$  for the special case where  $p_i / \alpha_i < p_j / \alpha_j$  implies  $p_i / \beta_i < p_j / \beta_j$  for all  $i$  and  $j$ .

For the general earliness-tardiness problems where no restrictions are placed on the values of  $\alpha_i$  and  $\beta_i$ ,

no optimization procedures are known to exist. Biskup and Feldmann [2] developed a problem generator and solve 280 instances with two new heuristics to identify upper bounds on the optimal function value. The first step of both heuristics assume that all jobs start on or after the common due date. In the following iterative steps, jobs are selected one by one by some criteria, and the selected jobs are moved so that they all start before the common due date. The iterative steps stop if some conditions met. One stopping condition is insufficient idle time to insert a job. However, if the optimal schedule has a straddling job, that is, if a job starts before the common due date, and completes after the common due date, their heuristics cannot find the optimal schedule.

Some studies employed meta-heuristic approaches, such as genetic algorithms and tabu search, to improve the quality of the solutions. Lee and Kim [14] proposed a parallel genetic algorithm, whereas James [6] developed a tabu search algorithm for solving general earliness-tardiness penalties about a common due date. Both approaches constrain their search spaces to schedules whose first job starts at time zero. This assumption may result in the exclusion of the optimal schedule. Examples of optimal schedules with the first job starting at time not equal to zero are presented in [17]. Feldmann and Biskup [5] applied five meta-heuristic approaches, namely evolutionary strategies, evolutionary search with a destabilization phase, simulated annealing, threshold accepting, and threshold accepting with a “back step” to obtaining near-optimal solutions by solving 140 benchmark problems. Hino et al. [9] proposed a tabu search-based heuristics, a genetic algorithm to minimize the total penalties, and hybrid strategies were also used to improve the performance of two methods. Meta-heuristic approaches may obtain near-optimal solutions; however, they are generally very time-consuming and may result in different solutions during different runs or attempts.

To obtain the optimal objective value more efficiently, three well-established optimality properties in common due-date scheduling are also employed in this study.

**Property 1:** An optimal schedule does not contain any idle time between any consecutive jobs.

Proof. This property is evident and can be established by contradiction; a detailed proof can be seen in [12].

Because of property 1, once the sequence of jobs is determined, the completion time for each job  $i$  equals the completion time of the previous job plus the processing time of job  $i$ . The start time of the first job can be computed as the time gap between the common due date and the summation of the processing time of the jobs starting before the common due date. If the time gap is negative, the start time of the first job is set at time zero.

**Property 2:** The optimal schedule is V-shaped around the common due-date. In the optimal schedule, the jobs completed before or on the common due date  $d$  are scheduled in non-increasing order of the ratios  $p_i/\alpha_i$ , and the jobs starting on or after  $d$  are scheduled in non-decreasing order of the ratios  $p_i/\beta_i$ .

Proof. The proof follows immediately from Smith's ratio rule [16], or can be proved by a job interchange argument [7, 8].

To enumerate the sequence of  $n$  jobs in the solution space, the number of the possible permutation is  $n$  factorial, which becomes elusive when  $n$  is large. By utilizing Property 2, jobs can be classified as starting either before or on/after the common due date. The solution space for the scheduling with binary coding of the jobs can be applied to decreasing the size of the solution space due to the V-shaped property. Thus, the number of possible sequences of jobs is reduced to  $2^n$ .

**Property 3:** In the optimal schedule, either the first job starts from time zero or the completion time of one job coincides with the common due date  $d$ .

Proof. The proof is available in [10].

In the proposed approach, the generated schedule follows Property 3 by setting the start time of the first job at time zero, or the completion time of a job coincident with the exact time of the common due

date. It is noted that setting the start time of the first job at time zero, a straddling job may exist in the resulting schedule.

The following section develops and elaborates the proposed sequential pair exchange approach by using the above well-established properties.

### 3. METHODOLOGY

The sequence pair exchange (denoted as SPE) approach is proposed in this paper. Since at most one job can be completed on the common due date, jobs can be divided into two sets  $S_E$  and  $S_L$ , where jobs in set  $S_E$  start before the common due date, and jobs in set  $S_L$  start on or after the common due date. A straddling job exists if one job starts before the common due date and completes after the common due date. The straddling job is in  $S_E$  as it has to start before the common due date. The proposed approach exchanges or shifts jobs between the two sets systematically to derive a near-optimal solution.

Three main steps are employed in SPE, and SPE can be classified into two types: SPE-1 and SPE-2. The only difference between SPE-1 and SPE-2 is the initial assignment.

In SPE-1, each job is considered for assignment to  $S_E$  or  $S_L$  simultaneously. The job with the smallest  $p_i / \alpha_i$  or  $p_i / \beta_i$  value has the highest priority to be assigned to one of the two sets. If the  $p_i / \alpha_i$  value of job  $i$  is smaller than the  $p_i / \beta_i$  value and the remaining time gap before the due date is not zero, job  $i$  is assigned to  $S_E$ ; otherwise, the job is assigned to  $S_L$ . If ties occur, the job with the smallest processing time is selected. When a tie occurs, the job with the smaller processing time is assigned to its corresponding set. If ties still exist with respect to the smallest processing time, the job with the smallest earliness penalty is chosen. Yet, if a tie exists for the smallest earliness penalty, the job with the smallest tardiness penalty is selected. Contrary to existing approaches such as the one proposed by Biskup and Feldman [2] where the procedure terminates if the remaining time gap is not long enough to accommodate the entire job. Although the available time gap is not long enough, the proposed approach continues to assign the job to  $S_E$  by generating a straddling job before terminating the procedure. After this initial assignment step is finished, jobs in  $S_E$  and  $S_L$  are sequenced according to the V-shaped property.

SPE-2 assigns jobs to  $S_E$  considering only the  $p_i / \alpha_i$  value of the job. The job with the smallest  $p_i / \alpha_i$  value has the highest priority to be assigned to  $S_E$ . If a tie occurs, the job with the smallest processing time, the smallest earliness penalty or the smallest tardiness penalty –evaluated in this order– will be chosen. Jobs are assigned to  $S_E$  one by one until the time gap equals zero or the number of jobs in  $S_E$  exceeds a predetermined integer value, computed as  $\lfloor n \times h \rfloor$ , where  $n$  denotes the number of jobs to be scheduled,  $h$  denotes the restrictive factor of the due date against the total processing time for determining the common due date, and  $\lfloor \cdot \rfloor$  denotes the largest integer smaller than or equal to the enclosed value. If the number of jobs in  $S_E$  will exceed the predetermined value or the time gap is down to zero, then all the remaining jobs are assigned to  $S_L$ . Likewise, jobs in  $S_E$  and  $S_L$  are sequenced according to the V-shaped property after finishing the initial assignment.

The second step of SPE is called the forward exchange. In this step, the jobs in  $S_E$  are selected one by one according to the sequence resulting from the first step. The selected  $i^{\text{th}}$  job is paired with all the jobs in  $S_L$  to identify the exchange which would lead to the largest improvement of the objective function value. Additionally, the  $k^{\text{th}}$  job in  $S_E$  may be shifted to  $S_L$  if this shift leads to a greater reduction of the objective function value than exchanging jobs. If exchanging  $i^{\text{th}}$  job in  $S_E$  with  $j^{\text{th}}$  job in  $S_L$  causes the best improvement of objective function value than shifting, the  $i^{\text{th}}$  job in  $S_E$  and  $j^{\text{th}}$  job in  $S_L$  are exchanged, while the original sequences in  $S_E$  and  $S_L$  are maintained. Conversely, if shifting  $k^{\text{th}}$  job in  $S_E$  to  $S_L$  results in a best improvement of the objective function value than exchanging, the  $k^{\text{th}}$  job is shifted to  $S_L$ . After examining all the jobs in  $S_E$  to be exchanged with jobs in  $S_L$ , or to be shifted to  $S_L$ , all jobs in  $S_E$  and  $S_L$  are sequenced according to the V-shaped property again.

The third step is called the backward exchange. This step is similar to the forward exchange step, except that jobs in  $S_L$  are selected one by one according to the sequence obtained from the second step, and jobs are exchanged in  $S_E$  to find the best improvement of the objective function value. Conversely, jobs in  $S_L$  are shifted to  $S_E$  if the best improvement of objective function value is attained. After examining all the jobs in  $S_L$  to be exchanged with job in  $S_E$  or shifted to  $S_E$ , all jobs in  $S_L$  and  $S_E$  are sequenced according to the V-shaped property again.

The detail description of SPE-1 and SPE-2 and the notation used in the proposed approach are described as follows.

| Notation         | Description                                                                                                                        |
|------------------|------------------------------------------------------------------------------------------------------------------------------------|
| $N$              | the number of jobs to be scheduled                                                                                                 |
| $\alpha_i$       | the penalty per unit time of earliness of job $i$                                                                                  |
| $\beta_i$        | the penalty per unit time of tardiness of job $i$                                                                                  |
| $p_i$            | the processing time of job $i$                                                                                                     |
| $D$              | the common due date                                                                                                                |
| $S_P$            | the set of jobs to be allocated                                                                                                    |
| $S_E$            | the set of jobs starting before the due date                                                                                       |
| $S_L$            | the set of jobs starting on or after the due date                                                                                  |
| $S_{E'}$         | the sorted set in which jobs are sequences in non-increasing order of ratio $p_i / \alpha_i$ and contains the same jobs as $S_E$ . |
| $S_{L'}$         | the sorted set in which jobs are sequences in non-decreasing order of ratio $p_i / \beta_i$ and contains the same jobs as $S_L$ .  |
| $S$              | the schedule representation $S = (S_{E'}, S_{L'})$                                                                                 |
| $f(S)$           | the objective function, which is $f(S) = \sum_{i=1}^n (\alpha_i E_i + \beta_i T_i)$ .                                              |
| $S_{t1}, S_{t2}$ | the temporary set                                                                                                                  |
| $E$              | the candidate job for $S_E$                                                                                                        |
| $L$              | the candidate job for $S_L$                                                                                                        |
| $N_{JobMax}$     | the maximum number of jobs admissible in $S_E$                                                                                     |
| $N_{JobE}$       | the number of jobs in $S_E$                                                                                                        |
| $T_g$            | the time gap for assigning the next job, $T_g = d - \sum_{i \in S_E} p_i$                                                          |
| $F_{Best}$       | the current best objective function value                                                                                          |
| $F_{Exchange}$   | the best objective function value obtained from job exchanges                                                                      |
| $F_{Shift}$      | the best objective function value obtained from job shifts                                                                         |
| $i, j, k$        | the index variable                                                                                                                 |
| $Index_1$        | the buffer for storing the index resulting in the best exchange improvement                                                        |
| $Index_2$        | the buffer for storing the index resulting in the best shift improvement                                                           |

### (1) SPE-1 Procedure:

#### Step 1 Initial Assignment

Step 1.1 Let  $S_P = \{1, 2, \dots, n\}$ ,  $S_E = S_L = \{\emptyset\}$ ,  $N_{JobE} = 0$ .

Step 1.2 Let  $E$  be the job that satisfies  $(p_E / \alpha_E) = \min_{i \in S_P} (p_i / \alpha_i)$  and  $L$  be the job that satisfies  $(p_L / \beta_L) = \min_{i \in S_P} (p_i / \beta_i)$ .

Step 1.3 If  $p_E / \alpha_E < p_L / \beta_L$  {  
     If  $T_g \geq p_E$  {



$S_E = S_E + \{E\}, S_P = S_P - \{E\}, N_{JobE} = N_{JobE} + 1$ , go to Step 1.2.  
 $\}$   
 If  $p_E > T_g$  (straddling job exists) {  
 $S_E = S_E + \{E\}, S_P = S_P - \{E\}, N_{JobE} = N_{JobE} + 1$ , go to Step 1.4.  
 $\}$   
 $\}$   
 Otherwise  
 $S_L = S_L + \{L\}, S_P = S_P - \{L\}$ , go to Step 1.2.  
 Step 1.4  $S_L = S_L \cup S_P$ .  
 Step 1.5  $S_E = S_E', S_L = S_L', f_{Best} = f(S)$ .

## Step 2 Forward Pair Exchange

### Step 2.1

For  $i=1$  to  $N_{JobE}$  {  
 $S_{t1} = S_E, S_{t2} = S_L, Index_1 = 0, Index_2 = 0$ .  
 For  $j=1$  to  $n - N_{JobE}$  {  
 $S_E = S_{t1}, S_L = S_{t2}$ , Exchange the  $i^{th}$  job in  $S_E$  with the  $j^{th}$  job in  $S_L, F_{Exchange} = f(S)$ .  
 If  $F_{Exchange} < F_{Best}$  {  
 $Index_1 = j, F_{Best} = F_{Exchange}$ .  
 $\}$   
 $\}$   
 For  $k=1$  to  $N_{JobE}$  {  
 $S_E = S_{t1}, S_L = S_{t2}$ , Shift the  $k^{th}$  job in  $S_E$  to  $S_L, F_{Shift} = f(S)$ .  
 If  $F_{Shift} < F_{Best}$  {  
 $Index_1 = 0, Index_2 = k, F_{Best} = F_{Shift}$ .  
 $\}$   
 $\}$   
 If  $Index_1 > 0$  {  
 $S_E = S_{t1}, S_L = S_{t2}$ , Exchange the  $i^{th}$  job in  $S_E$  with the  $Index_1^{th}$  job in  $S_L$ .  
 $\}$   
 If  $Index_1 = 0$  and  $Index_2 > 0$  {  
 $S_E = S_{t1}, S_L = S_{t2}$ , Shift the  $Index_2^{th}$  job in  $S_E$  to  $S_L, N_{JobE} = N_{JobE} - 1$ .  
 $\}$   
 $\}$

Step 2.2  $S_E = S_E', S_L = S_L', f_{Best} = f(S)$ .

## Step 3 Backward Pair Exchange

### Step 3.1

For  $j=1$  to  $n - N_{JobE}$  {  
 $S_{t1} = S_E, S_{t2} = S_L, Index_1 = 0, Index_2 = 0$ .  
 For  $i=1$  to  $N_{JobE}$  {  
 $S_E = S_{t1}, S_L = S_{t2}$ , Exchange the  $j^{th}$  job in  $S_L$  with the  $i^{th}$  job in  $S_E, F_{Exchange} = f(S)$ .  
 If  $F_{Exchange} < F_{Best}$  {  
 $Index_1 = i, F_{Best} = F_{Exchange}$ .  
 $\}$   
 $\}$   
 For  $k=1$  to  $n - N_{JobE}$  {  
 $S_E = S_{t1}, S_L = S_{t2}$ , Shift the  $k^{th}$  job in  $S_L$  to  $S_E, F_{Shift} = f(S)$ ,

```

    If  $F_{Shift} < F_{Best}$  {
         $Index_1 = 0, Index_2 = k, F_{Best} = F_{Shift}$ .
    }
}
If  $Index_1 > 0$  {
     $S_E = S_{t1}, S_L = S_{t2}$ , Exchange the  $j^{th}$  job in  $S_L$  with  $Index_1^{th}$  job in  $S_E$ .
}
If  $Index_1 = 0$  and  $Index_2 > 0$  {
     $S_E = S_{t1}, S_L = S_{t2}$ , Shift the  $Index_2^{th}$  job in  $S_L$  to  $S_E, N_{JobE} = N_{JobE} + 1$ .
}
}
Step 3.2  $S_E = S_{E'}, S_L = S_{L'}, f_{Best} = f(S)$ .

```

## (2) SPE-2 Procedure:

The procedure of the SPE-2 is much like that of the SPE-1, except for the initial assignment steps. The initial assignment is described as follows.

```

Step 1.1 Let  $S_p = \{1, 2, \dots, n\}, S_E = S_L = \{\emptyset\}, N_{JobE} = 0, N_{JobMax} = \lfloor n \times h \rfloor$ .
Step 1.2 Let  $E$  denotes the job that satisfies  $(P_E / \alpha_E) = \min_{i \in S_p} (P_i / \alpha_i)$ .
Step 1.3 If  $N_{JobE} < N_{JobMax}$  and  $T_g \geq P_E$  {
     $S_E = S_E + \{E\}, S_p = S_p - \{E\}, N_{JobE} = N_{JobE} + 1$ , go to Step 1.2.
}
If  $N_{JobE} < N_{JobMax}$  and  $P_E > T_g$  (straddling job exists) {
     $S_E = S_E + \{E\}, S_p = S_p - \{E\}, N_{JobE} = N_{JobE} + 1$ , go to Step 1.4.
}
Otherwise, go to Step 1.4.
Step 1.4  $S_L = S_L \cup S_p$ .
Step 1.5  $S_E = S_{E'}, S_L = S_{L'}, f_{Best} = f(S)$ .

```

When computing the objective function values, all jobs in  $S_E$  and  $S_L$  need to be sequenced according to the V-shaped property. The starting time of the first job is computed as  $d - \sum_{i \in S_E} p_i$ . If  $d - \sum_{i \in S_E} p_i < 0$ , the starting time of the first job is set to be zero. Evidently, if the period between time zero and the common due date is short respective to the total processing time, the starting time of the first job is more likely to be at time zero in an optimal schedule. Therefore, another way to computing the starting time of the first job is set the starting time being zero. After determining the starting time of the first job, the completed time of other jobs can be calculated easily according to their sequence on the schedule. And the smaller objective function value obtained is chosen as the total penalties for this schedule.

When the number of jobs starting before the common due date equals  $n_1$ , there will be  $n_1$  iterations in the forward exchange. In each iteration,  $n - n_1$  jobs are exchanged and  $n_1$  jobs are shifted to examine if there exists improvement on the objective function value. Thus  $n$  possible solutions are examined in each iteration. Therefore,  $n_1 n$  possible solutions with respect to various exchanges and shifts must be examined. Similarly, there will be  $n - n_1$  iterations in backward exchange. For each iteration,  $n_1$  jobs are exchanged and  $n - n_1$  jobs are shifted to determine if there exists improvement of the objective function value. Therefore, there are  $n$  possible solutions examined in one iteration. Thus,  $(n - n_1)n$  exchanges and shifts are to be examined in the backward exchange. Therefore, there are  $n^2$  jobs possible solutions to be examined in forward exchange and backward step. And because there may be two starting time of the first job in one

solution to compute the objective function values, which represents two schedules need to be examined. Therefore, the number of the possible schedules to be examined is at most  $2n^2$ .

#### 4. DEMONSTRATION OF THE PROCEDURE

The proposed approach is further elaborated by solving the example problem provided in [2] with the choice of  $k=9$  and  $n=10$ , that is, the ninth instance of the problem set with 10 jobs. The common due date is calculated as  $d = \lfloor h \sum p_i \rfloor$ , where  $h$  represents the restrictive factor of the due date against the total processing time and is set to be 0.4 and 0.8 for examples 1 and 2, respectively. Table 1 presents the processing time, earliness penalties and tardiness penalties of the case problem. Both examples for the case problem demonstrate how the proposed approach leads to an optimal solution.

Table 1. Case problem with  $n=10$ .

| Job        | 1  | 2 | 3  | 4  | 5  | 6 | 7  | 8  | 9  | 10 |
|------------|----|---|----|----|----|---|----|----|----|----|
| $p_i$      | 10 | 6 | 11 | 20 | 8  | 3 | 18 | 10 | 1  | 5  |
| $\alpha_i$ | 1  | 4 | 9  | 3  | 6  | 7 | 6  | 4  | 8  | 7  |
| $\beta_i$  | 2  | 6 | 3  | 14 | 14 | 7 | 10 | 6  | 15 | 13 |

##### Example 1 ( $h=0.4$ ):

In this example, with  $h=0.4$ , the common due date is set to be 36, and for demonstration purposes SPE-1 is utilized to solve the problem. After the initial assignment step, jobs in  $S_E$  and  $S_L$  are  $\{3,6\}$  and  $\{9,10,5,2,4,8,7,1\}$ , respectively, both of which conform to the V-shaped property. Therefore, the number of jobs in  $S_E$ ,  $N_{JobE}$  is 2, the current best schedule is  $\{\{3,6\}, \{9,10,5,2,4,8,7,1\}\}$  and the current best objective function value,  $F_{Best}$  is 1,707.

In the forward exchange step, two iterations are executed. In the iteration 1, exchanging job 3 with job 4 leads to the most objective function value improvement; therefore, jobs in  $S_E$  and  $S_L$  become  $\{4,6\}$  and  $\{9,10,5,2,3,8,7,1\}$ , respectively, after this exchange, and  $F_{Best}$  value is improved to 1,042, and the current best schedule is  $\{\{4,6\}, \{9,10,5,2,8,7,3,1\}\}$ . Notice that, when computing the objective function value, the schedule is constructed using  $S_{E'}$  and  $S_{L'}$  instead of  $S_E$  and  $S_L$ . In iteration 2, exchanging jobs 6 and 5 will leads to the best improvement of objective function value. The jobs in  $S_E$  and  $S_L$  become  $\{4,5\}$  and  $\{9,10,6,2,3,8,7,1\}$  after this exchange, and the  $F_{Best}$  value becomes 917, and the current best schedule is  $\{\{4,5\}, \{9,10,6,2,8,7,3,1\}\}$ . Then, all jobs in  $S_E$  and  $S_L$  are sequenced according to the V-shaped property, producing jobs in  $S_E$  and  $S_L$  being  $\{4,5\}$  and  $\{9,10,6,2,8,7,3,1\}$ , respectively. Thus, the current best schedule becomes  $\{\{4,5\}, \{9,10,6,2,8,7,3,1\}\}$ .

In the backward exchange step, eight iterations are executed. In iteration 1, shifting job 10 from  $S_L$  to  $S_E$  results in the greatest reduction of objective function value; therefore, job 10 is shifted to  $S_E$ , and jobs in  $S_E$  and  $S_L$  become  $\{4,5,10\}$  and  $\{9,6,2,8,7,3,1\}$ , respectively. After this iteration,  $N_{JobE}$  is increased to 3, the current best schedule is  $\{\{4,5,10\}, \{9,6,2,8,7,3,1\}\}$  and  $F_{Best}$  is reduced to 884. In iteration 2, shifting job 6 from  $S_L$  to  $S_E$  results in the greatest reduction of objective function value; therefore, job 9 is shifted to  $S_E$ , and jobs in  $S_E$  and  $S_L$  become  $\{4,5,10,6\}$  and  $\{9,2,8,7,3,1\}$ , respectively. After this iteration,  $N_{JobE}$  is increased to 4; the current best schedule is  $\{\{4,5,10,6\}, \{9,2,8,7,3,1\}\}$  and  $F_{Best}$  is reduced to 876. Because other iterations can not reduce the objective function value, the schedule remains the same as in iteration 2 after finishing eight iterations. Then, sequencing all jobs in  $S_E$  and  $S_L$  according to the V-shaped property again produces jobs in  $S_E$  and  $S_L$  being  $\{4,5,10,6\}$  and  $\{9,2,8,7,3,1\}$ , respectively, and the current best schedule becomes  $\{\{4,5,10,6\}, \{9,2,8,7,3,1\}\}$ , and the  $F_{Best}$  value being 876. The problem has been solved by using optimization software LINDO in [2]. The result obtained by the proposed approach in this paper is the same as the optimal obtained from LINDO.

**Example 2 ( $h=0.8$ ):**

In this example, with  $h=0.8$ , the common due date is set to be 73, and for the demonstration purposes SPE-2 is applied to solve the problem. After the initial assignment step, jobs in  $S_E$  and  $S_L$  are  $\{7,8,2,5,3,10,6,9\}$  and  $\{4,1\}$ , respectively. Thus, the number of jobs in  $S_E$ ,  $N_{JobE}$  is 8, the current best schedule is  $\{\{7,8,2,5,3,10,6,9\},\{4,1\}\}$  and the current best objective function value,  $F_{Best}$  is 1,088.

In the forward exchange and shift step, eight iterations are executed. In iteration 1, exchange job 7 and 4 will results in the greatest reduction of objective function value, therefore, job 7 and 4 are exchanged, and jobs in  $S_E$  and  $S_L$  become  $\{4,8,2,5,3,10,6,9\}$  and  $\{7,1\}$ , respectively. After this iteration, the current best schedule becomes  $\{\{4,8,2,5,3,10,6,9\},\{7,1\}\}$  and  $F_{Best}$  is reduced to 852. In the iteration 2, shifting job 3 from  $S_E$  to  $S_L$  results in the greatest reduction of objective function value; therefore, job 3 is shifted to  $S_L$ , and jobs in  $S_E$  and  $S_L$  become  $\{4,8,2,5,10,6,9\}$  and  $\{7,1,3\}$ , respectively. After this iteration,  $N_{JobE}$  is decreased to 7, the current best schedule is  $\{\{4,8,2,5,10,6,9\},\{7,3,1\}\}$  and  $F_{Best}$  is reduced to 693. In iteration 3, exchanging jobs 2 and 7 leads to the greatest reduction of objective function value; therefore, jobs 2 and 7 are exchanged, and jobs in  $S_E$  and  $S_L$  become  $\{4,8,7,5,10,6,9\}$  and  $\{2,1,3\}$ , respectively. After this iteration, the  $F_{Best}$  value is reduced to 595 and the current best schedule becomes  $\{\{4,7,8,5,10,6,9\},\{2,3,1\}\}$ . In iteration 4, shifting job 6 from  $S_E$  to  $S_L$  results in the greatest reduction of objective function value; therefore, job 6 is shifted to  $S_L$ , and jobs in  $S_E$  and  $S_L$  become  $\{4,8,7,5,10,9\}$  and  $\{2,1,3,6\}$ , respectively. After this iteration,  $N_{JobE}$  is decreased to 6, the current best schedule is  $\{\{4,7,8,5,10,9\},\{6,2,3,1\}\}$  and  $F_{Best}$  is reduced to 564. Because the remaining iterations cannot improve the objective function value, the jobs in  $S_E$  and  $S_L$  remain as those in iteration 4. After finishing eight iterations, sequencing all jobs in  $S_E$  and  $S_L$  according to the V-shaped property again produces jobs in  $S_E$  and  $S_L$  being  $\{4,8,7,5,10,9\}$  and  $\{6,2,3,1\}$ , respectively, and the  $F_{Best}$  value being 564, and the current best schedule becomes  $\{\{4,7,8,5,10,9\},\{6,2,3,1\}\}$ .

In the backward exchange step, only in the iteration 4, exchange job 1 and 8 will results in the greatest reduction of objective function value, therefore, job 1 and 8 are exchanged, and jobs in  $S_E$  and  $S_L$  become  $\{4,7,1,5,10,9\}$  and  $\{6,2,3,8\}$ , respectively. After this iteration, the current best schedule becomes  $\{\{1,4,7,5,10,9\},\{6,2,8,3\}\}$  and  $F_{Best}$  is reduced to 554. After four iterations, sequencing all jobs in  $S_E$  and  $S_L$  according to the V-shaped property again produces jobs in  $S_E$  and  $S_L$  being  $\{1,4,7,5,10,9\}$  and  $\{6,2,8,3\}$ , respectively, and the current best schedule becomes  $\{\{1,4,7,5,10,9\},\{6,2,8,3\}\}$ , and the  $F_{Best}$  value being 554. Again, this solution is the same as the one computed by using LINDO in [2] and thus is the optimum.

**5. EXPERIMENTAL RESULTS**

To illustrate the effectiveness of the proposed approach by comparing against the existing approach, problem sets from Biskup and Feldmann [2] were adopted for experiment. The proposed approach was implemented in the C language and run on a Pentium III-1.0G machine with 256 MB of RAM. The number of jobs  $n$  is set to be 10, 20, 50, 100, 200, 500 and 1000. The restrictive factor  $h$  for determining the common due date  $d$ , which can be computed as  $d = \lfloor h \sum p_i \rfloor$ , is set to be 0.2, 0.4, 0.6 and 0.8. Ten problems are examined for each combination of the number of jobs  $n$  and the restrictive factor  $h$ . The value of  $k$  represents for the  $k^{\text{th}}$  instance of each problem set. Therefore, the proposed approach is applied to solve 280 problems. All Problems are solved by both SPE-1 and SPE-2, and the better result of SPE-1 and SPE-2 is taken as the objective function value for each problem. The best objective function values for 280 instances yielded by SPE approach are shown in Tables 2 to 8. They are the upper bounds on the optimal objective function for the particular instance. We label them 1, 2 in order indicate by which of the SPE-1 or SPE-2 approach the particular upper bounds obtained. If both SPE-1 and SPE-2 approach obtain the particular upper bounds, we label them by asterisk sign.

Table 2. Upper bounds found by the proposed approach for the 10-job cases.

| $n=10$ | $h=0.2$            | $h=0.4$            | $h=0.6$          | $h=0.8$ |
|--------|--------------------|--------------------|------------------|---------|
| $k=1$  | 1,936*             | 1,025*             | 841*             | 818*    |
| $k=2$  | 1,042*             | 615*               | 615*             | 615*    |
| $k=3$  | 1,586*             | 917 <sup>2</sup>   | 793*             | 793*    |
| $k=4$  | 2,139*             | 1,230*             | 815*             | 803*    |
| $k=5$  | 1,187 <sup>1</sup> | 630*               | 521*             | 521*    |
| $k=6$  | 1,523*             | 908*               | 755*             | 755*    |
| $k=7$  | 2,170*             | 1,374*             | 1,101*           | 1,083*  |
| $k=8$  | 1,720*             | 1,020*             | 610*             | 540*    |
| $k=9$  | 1,574*             | 876*               | 582*             | 554*    |
| $k=10$ | 1,869*             | 1,136 <sup>1</sup> | 711 <sup>2</sup> | 671*    |

Table 3. Upper bounds found by the proposed approach for the 20-job cases.

| $n=20$ | $h=0.2$             | $h=0.4$            | $h=0.6$            | $h=0.8$            |
|--------|---------------------|--------------------|--------------------|--------------------|
| $k=1$  | 4,394 <sup>2</sup>  | 3,066 <sup>2</sup> | 2,986 <sup>2</sup> | 2,987*             |
| $k=2$  | 8,430 <sup>2</sup>  | 4,847*             | 3,220*             | 2,980*             |
| $k=3$  | 6,210 <sup>1</sup>  | 3,838*             | 3,583*             | 3,583*             |
| $k=4$  | 9,188 <sup>1</sup>  | 5,118*             | 3,317 <sup>2</sup> | 3,040*             |
| $k=5$  | 4,215 <sup>1</sup>  | 2,495 <sup>2</sup> | 2,173*             | 2,173*             |
| $k=6$  | 6,527*              | 3,582*             | 3,016*             | 3,016 <sup>1</sup> |
| $k=7$  | 10,455 <sup>2</sup> | 6,272*             | 4,134*             | 3,878*             |
| $k=8$  | 3,920 <sup>2</sup>  | 2,145*             | 1,638*             | 1,638 <sup>1</sup> |
| $k=9$  | 3,485 <sup>1</sup>  | 2,096 <sup>2</sup> | 1,965*             | 1,965 <sup>1</sup> |
| $k=10$ | 4,979 <sup>2</sup>  | 2,925 <sup>1</sup> | 2,110 <sup>2</sup> | 1,995*             |

Table 4. Upper bounds found by the proposed approach for the 50-job cases.

| $n=50$ | $h=0.2$             | $h=0.4$             | $h=0.6$             | $h=0.8$             |
|--------|---------------------|---------------------|---------------------|---------------------|
| $k=1$  | 40,742 <sup>1</sup> | 23,941*             | 17,989 <sup>1</sup> | 17,934*             |
| $k=2$  | 30,769 <sup>2</sup> | 17,961*             | 14,053 <sup>2</sup> | 14,040*             |
| $k=3$  | 34,442 <sup>1</sup> | 20,530*             | 16,497*             | 16,497*             |
| $k=4$  | 27,755 <sup>1</sup> | 16,727 <sup>2</sup> | 14,080 <sup>2</sup> | 14,080 <sup>2</sup> |
| $k=5$  | 32,315 <sup>1</sup> | 18,055*             | 14,605*             | 14,605 <sup>1</sup> |
| $k=6$  | 34,969 <sup>1</sup> | 20,456 <sup>2</sup> | 14,251 <sup>1</sup> | 14,066*             |
| $k=7$  | 43,134 <sup>1</sup> | 23,038 <sup>2</sup> | 17,616 <sup>1</sup> | 17,616*             |
| $k=8$  | 43,840 <sup>2</sup> | 24,910*             | 21,329 <sup>1</sup> | 21,329*             |
| $k=9$  | 34,264 <sup>1</sup> | 20,047*             | 14,202 <sup>2</sup> | 13,942 <sup>1</sup> |
| $k=10$ | 33,105 <sup>1</sup> | 19,167 <sup>1</sup> | 14,380*             | 14,363*             |

Table 5. Upper bounds found by the proposed approach for the 100-job cases.

| $n=100$ | $h=0.2$              | $h=0.4$             | $h=0.6$             | $h=0.8$             |
|---------|----------------------|---------------------|---------------------|---------------------|
| $k=1$   | 145,634 <sup>1</sup> | 86,054 <sup>2</sup> | 72,017 <sup>1</sup> | 72,017*             |
| $k=2$   | 124,999 <sup>1</sup> | 72,998 <sup>2</sup> | 59,231*             | 59,231*             |
| $k=3$   | 129,911 <sup>2</sup> | 79,739 <sup>1</sup> | 68,537 <sup>1</sup> | 68,537 <sup>1</sup> |
| $k=4$   | 129,586 <sup>2</sup> | 79,405 <sup>2</sup> | 68,759 <sup>2</sup> | 68,759 <sup>2</sup> |
| $k=5$   | 124,390 <sup>1</sup> | 71,361 <sup>1</sup> | 55,286 <sup>2</sup> | 55,103*             |
| $k=6$   | 139,285 <sup>1</sup> | 77,986*             | 62,399*             | 62,399 <sup>1</sup> |
| $k=7$   | 135,274 <sup>1</sup> | 78,244*             | 62,197*             | 62,197*             |
| $k=8$   | 160,174 <sup>2</sup> | 94,397 <sup>2</sup> | 80,708 <sup>1</sup> | 80,708 <sup>1</sup> |
| $k=9$   | 116,551 <sup>2</sup> | 69,458 <sup>2</sup> | 58,727 <sup>1</sup> | 58,727 <sup>1</sup> |
| $k=10$  | 119,152 <sup>1</sup> | 71,921*             | 61,361 <sup>1</sup> | 61,361 <sup>1</sup> |

Table 6. Upper bounds found by the proposed approach for the 200-job cases.

| $n=200$ | $h=0.2$              | $h=0.4$              | $h=0.6$              | $h=0.8$              |
|---------|----------------------|----------------------|----------------------|----------------------|
| $k=1$   | 498,664 <sup>2</sup> | 295,777*             | 254,259*             | 254,259 <sup>1</sup> |
| $k=2$   | 541,184 <sup>1</sup> | 319,268 <sup>2</sup> | 266,003 <sup>1</sup> | 266,003*             |
| $k=3$   | 488,785 <sup>1</sup> | 293,886 <sup>2</sup> | 254,476*             | 254,476*             |
| $k=4$   | 586,279 <sup>1</sup> | 353,059 <sup>1</sup> | 297,109*             | 297,109 <sup>1</sup> |
| $k=5$   | 513,334 <sup>1</sup> | 304,733 <sup>2</sup> | 260,278 <sup>1</sup> | 260,278 <sup>1</sup> |
| $k=6$   | 478,056 <sup>2</sup> | 280,024 <sup>1</sup> | 235,707 <sup>1</sup> | 235,707 <sup>1</sup> |
| $k=7$   | 454,857 <sup>2</sup> | 275,131 <sup>1</sup> | 246,307*             | 246,307 <sup>1</sup> |
| $k=8$   | 494,568 <sup>2</sup> | 279,298 <sup>1</sup> | 225,215 <sup>2</sup> | 225,224 <sup>2</sup> |
| $k=9$   | 529,372 <sup>1</sup> | 310,658*             | 254,637*             | 254,637 <sup>1</sup> |
| $k=10$  | 538,561 <sup>2</sup> | 323,108 <sup>1</sup> | 268,354*             | 268,354 <sup>1</sup> |

Table 7. Upper bounds found by the proposed approach for the 500-job cases.

| $n=500$ | $h=0.2$                | $h=0.4$                | $h=0.6$                | $h=0.8$                |
|---------|------------------------|------------------------|------------------------|------------------------|
| $k=1$   | 2,954,981 <sup>2</sup> | 1,788,627 <sup>2</sup> | 1,579,035 <sup>2</sup> | 1,579,062 <sup>1</sup> |
| $k=2$   | 3,365,948 <sup>2</sup> | 1,995,145 <sup>1</sup> | 1,712,209 <sup>2</sup> | 1,712,206 <sup>2</sup> |
| $k=3$   | 3,102,914 <sup>2</sup> | 1,864,675 <sup>1</sup> | 1,641,438 <sup>2</sup> | 1,641,457 <sup>1</sup> |
| $k=4$   | 3,221,043 <sup>2</sup> | 1,887,522 <sup>2</sup> | 1,640,787 <sup>2</sup> | 1,640,814 <sup>1</sup> |
| $k=5$   | 3,114,922 <sup>2</sup> | 1,807,071 <sup>2</sup> | 1,468,231 <sup>1</sup> | 1,468,231 <sup>1</sup> |
| $k=6$   | 2,792,302 <sup>2</sup> | 1,610,120 <sup>1</sup> | 1,411,830 <sup>1</sup> | 1,411,830 <sup>1</sup> |
| $k=7$   | 3,172,463 <sup>2</sup> | 1,902,782 <sup>1</sup> | 1,634,330*             | 1,634,332 <sup>1</sup> |
| $k=8$   | 3,122,308 <sup>1</sup> | 1,819,313 <sup>1</sup> | 1,540,386 <sup>2</sup> | 1,540,377 <sup>2</sup> |
| $k=9$   | 3,364,449 <sup>2</sup> | 1,973,924 <sup>1</sup> | 1,680,188 <sup>2</sup> | 1,680,202*             |
| $k=10$  | 3,120,690 <sup>2</sup> | 1,837,583 <sup>1</sup> | 1,519,191*             | 1,519,191 <sup>1</sup> |

Table 8. Upper bounds found by the proposed approach for the 1000-job cases.

| $n=1000$ | $h=0.2$                 | $h=0.4$                | $h=0.6$                | $h=0.8$                |
|----------|-------------------------|------------------------|------------------------|------------------------|
| $k=1$    | 14,055,246 <sup>2</sup> | 8,111,053 <sup>1</sup> | 6,410,886 <sup>2</sup> | 6,410,915 <sup>1</sup> |
| $k=2$    | 12,296,235 <sup>2</sup> | 7,271,744 <sup>2</sup> | 6,110,092 <sup>2</sup> | 6,110,092 <sup>2</sup> |
| $k=3$    | 11,967,436 <sup>2</sup> | 6,986,953 <sup>2</sup> | 5,983,310 <sup>2</sup> | 5,983,313 <sup>1</sup> |
| $k=4$    | 11,796,911 <sup>1</sup> | 7,024,128 <sup>2</sup> | 6,085,884 <sup>2</sup> | 6,085,884 <sup>2</sup> |
| $k=5$    | 12,449,674 <sup>2</sup> | 7,364,863 <sup>2</sup> | 6,341,477 <sup>2</sup> | 6,341,481 <sup>1</sup> |
| $k=6$    | 11,644,172 <sup>2</sup> | 6,927,792 <sup>2</sup> | 6,078,383 <sup>1</sup> | 6,078,383 <sup>1</sup> |
| $k=7$    | 13,277,161 <sup>2</sup> | 7,861,782 <sup>2</sup> | 6,574,305 <sup>2</sup> | 6,574,297 <sup>2</sup> |
| $k=8$    | 12,274,824 <sup>2</sup> | 7,222,199 <sup>2</sup> | 6,067,312 <sup>2</sup> | 6,067,324 <sup>2</sup> |
| $k=9$    | 11,757,145 <sup>2</sup> | 7,058,879 <sup>2</sup> | 6,185,335 <sup>2</sup> | 6,185,349 <sup>1</sup> |
| $k=10$   | 12,427,705 <sup>2</sup> | 7,276,070 <sup>2</sup> | 6,145,750 <sup>2</sup> | 6,145,737 <sup>2</sup> |

It is noted, for the benchmark problem with  $n=10$ , most solutions obtained from the proposed approach are as good as those optimal solutions identified by using software packages such as LINDO [2]. In other words, the proposed approach can lead to optimal solutions in most of the experimental cases.

To evaluate the results, the percentage improvement of the obtained solution value ( $F_{SPE}$ ) was compared with regard to the benchmark values provided by Biskup and Feldman ( $F_{BF}$ ), which can be computed as follows:

$$Improvement\ rate = \frac{F_{BF} - F_{SPE}}{F_{BF}} * 100\%$$

Table 9 lists the results obtained from SPE. Each cell of the table denotes the average improvement for the 10 instances under a number of jobs  $n$  and a restrictive factor  $h$ . Analysis of the average value with respect to the number of jobs  $n$  and restrictive factor  $h$  indicates that the SPE approach produces better results than the benchmark values for most of the problems. The average improvement rates are 5.794%,

3.59%, 0.204%, and 0.182% for  $h=0.2$ , 0.4, 0.6, and 0.8, respectively. For  $h=0.2$  and  $h=0.4$ , the results indicate a larger improvement, whereas for  $h=0.6$  and  $h=0.8$ , the results show a smaller improvement.

Table 9. Improvement rate (%) of SPE over other dedicated heuristic approaches.

| $h$     | $n=10$ | $n=20$ | $n=50$ | $n=100$ | $n=200$ | $n=500$ | $n=1000$ | Average |
|---------|--------|--------|--------|---------|---------|---------|----------|---------|
| 0.2     | 6.114  | 3.785  | 5.580  | 6.121   | 5.759   | 6.428   | 6.771    | 5.794   |
| 0.4     | 2.166  | 1.577  | 4.420  | 4.852   | 3.725   | 3.568   | 4.397    | 3.529   |
| 0.6     | 0.009  | 0.639  | 0.313  | 0.145   | 0.154   | 0.112   | 0.058    | 0.204   |
| 0.8     | 0.147  | 0.385  | 0.242  | 0.176   | 0.154   | 0.111   | 0.057    | 0.182   |
| Average | 2.109  | 1.597  | 2.639  | 2.824   | 2.448   | 2.555   | 2.281    | 2.427   |

To gain further insight on the performance of the proposed approach, additional analyses are shown in Tables 10 and 11 for SPE-1 and SPE-2. The number of successes is defined as the total number of problems that the best solution results from the procedure/approach in question. The number of ties denotes the number of problems in which the best solution resulting from both of the analyzed approach having the same objective function value. Notably, Compared to the benchmark problems when  $h=0.2$  or  $h=0.4$ , the SPE achieved better results; that is, having better objective function values, in 95.71% of the instances, equal results in 4.29% of the instances, and no worse results of the instances. On the other hand, compared with benchmark problems when  $h=0.6$  or  $h=0.8$ , the SPE achieved better results in 77.14% of the instances, equal results in 21.43% of instances, and worse results in 1.43% of the instances. This is to be expected as with the common due date being later, the particular handling of the straddling jobs in the proposed approach would not be as effective as for problems with earlier common due date.

Table 10. Comparison between SPE and other dedicated heuristic when  $h=0.2$  and  $h=0.4$ .

| Number of jobs $n$             | 10  |     | 20  |     | 50  |     | 100 |     | 200 |     | 500 |     | 1,000 |     | Total      |
|--------------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|-----|------------|
| Restrictive factor $h$         | 0.2 | 0.4 | 0.2 | 0.4 | 0.2 | 0.4 | 0.2 | 0.4 | 0.2 | 0.4 | 0.2 | 0.4 | 0.2   | 0.4 |            |
| Number of success of SPE-1     | 10  | 5   | 10  | 7   | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10    | 10  | <b>132</b> |
| Number of success of benchmark | 0   | 1   | 0   | 2   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 3          |
| Number of Ties                 | 0   | 4   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 5          |
| Number of success of SPE-2     | 10  | 6   | 9   | 9   | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10    | 10  | <b>134</b> |
| Number of success of benchmark | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 1          |
| Number of Ties                 | 0   | 4   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 5          |
| Number of success of SPE       | 10  | 6   | 10  | 9   | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10    | 10  | <b>134</b> |
| Number of success of benchmark | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 0          |
| Number of Ties                 | 0   | 4   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 6          |

Table 11. Comparison between SPE and other dedicated heuristic when  $h=0.6$  and  $h=0.8$ .

| Number of jobs $n$             | 10  |     | 20  |     | 50  |     | 100 |     | 200 |     | 500 |     | 1,000 |     | Total      |
|--------------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|-----|------------|
| Restrictive factor $h$         | 0.6 | 0.8 | 0.6 | 0.8 | 0.6 | 0.8 | 0.6 | 0.8 | 0.6 | 0.8 | 0.6 | 0.8 | 0.6   | 0.8 |            |
| Number of success of SPE-1     | 1   | 1   | 6   | 4   | 6   | 9   | 9   | 9   | 10  | 10  | 10  | 10  | 10    | 10  | <b>105</b> |
| Number of success of benchmark | 1   | 0   | 2   | 1   | 2   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 6          |
| Number of Ties                 | 8   | 9   | 2   | 5   | 2   | 1   | 1   | 1   | 0   | 0   | 0   | 0   | 0     | 0   | 29         |
| Number of success of SPE-2     | 1   | 1   | 7   | 4   | 6   | 8   | 8   | 9   | 10  | 9   | 10  | 10  | 10    | 10  | <b>103</b> |
| Number of success of benchmark | 0   | 0   | 0   | 3   | 3   | 0   | 2   | 1   | 0   | 1   | 0   | 0   | 0     | 0   | 10         |
| Number of Ties                 | 9   | 9   | 3   | 3   | 1   | 2   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 27         |
| Number of success of SPE       | 1   | 1   | 7   | 5   | 7   | 9   | 9   | 9   | 10  | 10  | 10  | 10  | 10    | 10  | <b>108</b> |
| Number of success of benchmark | 0   | 0   | 0   | 1   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0     | 0   | 2          |
| Number of Ties                 | 9   | 9   | 3   | 4   | 2   | 1   | 1   | 1   | 0   | 0   | 0   | 0   | 0     | 0   | 30         |

In order to demonstrate the superiority of the proposed approach, the result of the developed approach was also compared with meta-heuristic approaches by Feldmann and Biskup [5]. Feldmann and Biskup investigated the same benchmark problems by applying five different meta-heuristics, but only considered

the restrictive factor  $h$  being equal to 0.2 and 0.4. Each meta-heuristics was performed three times with different initial random numbers on a Pentium 90 PC. Therefore, each instance was solved 15 runs. The best objective function value obtained in the 15 runs was taken as the new upper bound for the benchmark problem. Hino et al. [9] proposed approaches, including tabu search (TS), genetic algorithm (GA), hybrid of tabu search and genetic algorithm (HTG) and hybrid of genetic algorithm and tabu search (HGT) to the same benchmark problems with restrictive factor  $h$  being equal to 0.2, 0.4, 0.6 and 0.8. Each instance was solved ten runs on a Pentium IV with a 1.7 GHz processor, and only the best objective function value of the ten runs was considered.

The results obtained from SPE are compared with meta-heuristic approaches are presented in Tables 12 and 13. The average improvement rates of meta-heuristics by Feldmann and Biskup and SPE against benchmark values are shown in Table 12. It is noted that the achieved improvement between the two approaches are very similar. Meta-heuristics would in general require more iterations and different runs with different random seeds, should lead to better solutions at the expense of time and stability of the solution. Yet, some average improvement rates of SPE are better than those of meta-heuristics by Feldmann and Biskup, as indicated by values marked with the asterisk sign in Table 12. A possible reason is that when the number of jobs increases, the size of the solution space increases exponentially. As such, meta-heuristics may not be able to explore a sufficient amount of trials to obtain a significantly good solution. For a thorough comparison of the proposed approach with other existing approaches, readers may refer to the web site at: <http://pc33.mis.hfu.edu.tw/SMGWET/>, for the execution of the proposed heuristic. The proposed approach demonstrates strong performance compared with the meta-heuristic approach as the differences between them are rather small.

Table 12. Average improvement rate (%) of meta-heuristics by Feldmann and Biskup and SPE against benchmark values.

| $n$     | $h$ | Feldmann and Biskup's meta-heuristics | SPE    |
|---------|-----|---------------------------------------|--------|
| 10      | 0.2 | 6.19                                  | 6.114  |
|         | 0.4 | 2.17                                  | 2.166  |
| 20      | 0.2 | 3.84                                  | 3.785  |
|         | 0.4 | 1.63                                  | 1.677  |
| 50      | 0.2 | 5.65                                  | 5.580  |
|         | 0.4 | 4.64                                  | 4.420  |
| 100     | 0.2 | 6.18                                  | 6.121  |
|         | 0.4 | 4.94                                  | 4.852  |
| 200     | 0.2 | 5.73                                  | 5.759* |
|         | 0.4 | 3.79                                  | 3.726  |
| 500     | 0.2 | 6.40                                  | 6.428* |
|         | 0.4 | 3.52                                  | 3.568* |
| 1000    | 0.2 | 6.72                                  | 6.771* |
|         | 0.4 | 4.30                                  | 4.397* |
| Average |     | 4.69                                  | 4.669  |

Table 13 shows the improvement rates of the meta-heuristic approach adopted by Hino et al. and our proposed approach. For  $h=0.2$  and 0.4, it can be observed that the solutions obtained by the proposed approach are very close to those by the four meta-heuristics, namely TS, GA, HTG and HGT approaches. Four out of fourteen improvement rates obtained by the proposed heuristic are better than those of the four meta-heuristics, as indicated by the values marked with an asterisk sign in Table 13. For  $h=0.6$  and  $h=0.8$ , when the number of jobs  $n$  is greater than or equal to 100, the average improvement rates of the proposed heuristic are better than or equal to those of all of the four meta-heuristic approaches. It is also noted that the solutions obtained by the four meta-heuristic are sometimes worse than the benchmark values obtained by a dedicated heuristic, as shown by the values with a negative sign in Table 13. This further supports our conjecture of why our heuristic can achieve better improvement results than the meta-heuristics do.



Table 13. Average improvement rate (%) of GA, TS, HTG, HGT and SEA against benchmark values.

| $N$     | $h$ | TS    | GA    | HTG   | HGT   | SEA   |
|---------|-----|-------|-------|-------|-------|-------|
| 10      | 0.2 | 5.95  | 6.07  | 6.07  | 6.07  | 6.09* |
|         | 0.4 | 1.93  | 1.98  | 1.98  | 1.98  | 2.15* |
|         | 0.6 | -0.08 | -0.00 | -0.00 | -0.00 | 0.01* |
|         | 0.8 | 0.15  | 0.15  | 0.15  | 0.15  | 0.15  |
| 20      | 0.2 | 3.84  | 3.84  | 3.84  | 3.84  | 3.84  |
|         | 0.4 | 1.62  | 1.62  | 1.62  | 1.62  | 1.57  |
|         | 0.6 | 0.71  | 0.68  | 0.71  | 0.71  | 0.64  |
|         | 0.8 | 0.41  | 0.28  | 0.41  | 0.41  | 0.40  |
| 50      | 0.2 | 5.70  | 5.68  | 5.70  | 5.70  | 5.60  |
|         | 0.4 | 4.66  | 4.60  | 4.66  | 4.66  | 4.37  |
|         | 0.6 | 0.32  | 0.31  | 0.27  | 0.31  | 0.31  |
|         | 0.8 | 0.24  | 0.19  | 0.23  | 0.23  | 0.24* |
| 100     | 0.2 | 6.19  | 6.17  | 6.19  | 6.19  | 6.16  |
|         | 0.4 | 4.93  | 4.91  | 4.93  | 4.93  | 4.75  |
|         | 0.6 | 0.01  | 0.12  | -0.08 | -0.04 | 0.15* |
|         | 0.8 | 0.15  | 0.12  | 0.08  | 0.11  | 0.18* |
| 200     | 0.2 | 5.76  | 5.74  | 5.76  | 5.76  | 5.75  |
|         | 0.4 | 3.74  | 3.75  | 3.75  | 3.75  | 3.67  |
|         | 0.6 | 0.01  | 0.13  | -0.37 | -0.07 | 0.15* |
|         | 0.8 | 0.04  | 0.14  | -0.26 | -0.07 | 0.15* |
| 500     | 0.2 | 6.41  | 6.41  | 6.41  | 6.41  | 6.42* |
|         | 0.4 | 3.57  | 3.58  | 3.58  | 3.58  | 3.36  |
|         | 0.6 | -0.25 | 0.11  | -0.73 | -0.15 | 0.11  |
|         | 0.8 | -0.21 | 0.11  | -0.73 | -0.13 | 0.11  |
| 1000    | 0.2 | 6.73  | 6.75  | 6.74  | 6.74  | 6.76* |
|         | 0.4 | 4.39  | 4.40  | 4.39  | 4.39  | 4.38  |
|         | 0.6 | -1.01 | 0.05  | -1.28 | -0.42 | 0.06* |
|         | 0.8 | -1.13 | 0.05  | -1.28 | -0.40 | 0.06* |
| Average |     | 2.31  | 2.43  | 2.24  | 2.37  | 2.41  |

The reason of using less powerful computer in this research is to demonstrate the efficiency of the proposed heuristic and to benchmark against existing papers with equivalent computing resources. The average running time of the proposed approach is presented in Table 14. When the number of jobs is below 500, the average running time is relatively short. The SPE was run on a Pentium III 1.0G PC, which is approximately eleven times faster than a Pentium 90 PC. Even after multiplying eleven times the average running time of SPE for different numbers of jobs, the running time required is still much smaller than those of the meta-heuristic approaches [5] running on a Pentium 90 PC.

Table 14. Running times (seconds) on a Pentium III 1.0G PC.

|         |       | $n=10$ | $n=20$ | $n=50$ | $n=100$ | $n=200$ | $n=500$ | $n=1,000$ |
|---------|-------|--------|--------|--------|---------|---------|---------|-----------|
| $h=0.2$ | SPE-1 | 0.001  | 0.002  | 0.011  | 0.045   | 0.253   | 3.645   | 28.771    |
|         | SPE-2 | 0.001  | 0.002  | 0.012  | 0.044   | 0.257   | 3.625   | 27.923    |
| $h=0.4$ | SPE-1 | 0.001  | 0.002  | 0.012  | 0.051   | 0.284   | 3.985   | 31.665    |
|         | SPE-2 | 0.001  | 0.002  | 0.012  | 0.052   | 0.267   | 3.856   | 30.063    |
| $h=0.6$ | SPE-1 | 0.001  | 0.002  | 0.011  | 0.045   | 0.234   | 3.354   | 27.089    |
|         | SPE-2 | 0.001  | 0.002  | 0.011  | 0.046   | 0.239   | 3.215   | 25.366    |
| $h=0.8$ | SPE-1 | 0.001  | 0.002  | 0.011  | 0.045   | 0.233   | 3.356   | 26.718    |
|         | SPE-2 | 0.001  | 0.002  | 0.010  | 0.045   | 0.229   | 3.213   | 25.116    |
| Average | SPE-1 | 0.001  | 0.002  | 0.011  | 0.047   | 0.251   | 3.585   | 28.561    |
|         | SPE-2 | 0.001  | 0.002  | 0.011  | 0.047   | 0.248   | 3.477   | 27.117    |

Based on the results obtained and the time required to run, the proposed approach shows the superiority in solving the common due-date problems with general earliness-tardiness penalties.

## 6. CONCLUSIONS AND FUTURE RESEARCHES

In this study, we propose a sequential pair exchange approach for solving the common due-date scheduling problem with general earliness-tardiness penalties. Computational results show that the proposed approach performs extremely well by finding lower total cost, and thus new upper bounds for most of the benchmark problems solved by the dedicated heuristics. Notably that, when compared with meta-heuristic approaches, the proposed approach also identifies new upper bounds for some cases, and the objective function values obtained are very close. The computing time required by the proposed algorithm is also relatively short when the number of jobs is smaller than 500. The experimental results indicate that the proposed approach can be applied to efficiently schedule common due-date problems with general earliness-tardiness penalties of a realistic size.

Possible directions for future research include utilizing the exchange and shift procedure into meta-heuristics, such as Simulated Annealing (SA), Genetic Algorithm (GA), Tabu Search (TS), and Ant Colony Optimization (ACO), and extending the proposed approach to solving the multi-machine scheduling problem with general or non-linear earliness-tardiness penalties about a common due date.

## ACKNOWLEDGEMENTS

The authors are grateful to Disk Biskup and Martin Feldman for making their benchmark problems available to this research.

## REFERENCE

1. Baker, K.R. and Scudder, G.D. (1990), Sequencing with earliness and tardiness penalties: a review, *Operations Research*, **38**, 1, 22-36.
2. Biskup, D. and Feldmann, M. (2001), Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers and Operations Research*, **28**, 8, 787-801.
3. Cheng, T.C.E. and Gupta, M.C. (1989), Survey of scheduling research involving due date determination decisions, *Europe Journal of Operational and Research*, **38**, 2, 156-166.
4. Gordon, V., Proth, J.M. and Chu, C. (2002), A survey of the state-of-art of common due date assignment and scheduling research, *European Journal of Operational Research*, **139**, 1, 1-25.
5. Feldmann, D. and Biskup, D. (2003), Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approach, *Computers and Industrial Engineering*, **44**, 2, 307-323.
6. James, R.J.W. (1997), Using tabu search to solve the common due date early/tardy machine scheduling problem, *Computers and Operations Research*, **24**, 3, 199-208.
7. Hall, N. G., Kubiak, W. and Sethi, S.P. (1991), Earliness-tardiness scheduling problem, II: Deviation of completion times about a restrictive common due date, *Operations Research*, **39**, 5, 847-856.
8. Hall, N.G. and Posner, M.E. (1991), Earliness-tardiness scheduling problem, I: Weighted deviation of completion times about a common due date, *Operations Research*, **39**, 5, 836-846.
9. Hino, C. M., Ronconi, D. P. and Mendes, A. B. (2005), Minimizing earliness and tardiness penalties in a single-machine problem with a common due date, *European Journal of Operational Research*, **160**, 1, pp. 190–201.
10. Hoogeveen, J.A. and van de Velde, S.L. (1991), Scheduling around a small common due date, *European Journal and Operational Research*, **55**, 2, 237-242.
11. Kanet, J.J. (1981), Minimizing the average deviation of job completion times about a common due date, *Naval Research Logistics Quarterly*, **28**:643-651.

12. Kahlbacher, H.G. (1993), Scheduling with monotonous earliness and tardiness penalties, *European Journal of Operational Research*, **64**, 2, 258-277.
13. Lee, C.Y., Danusaputro, S.L. and Lin, C.S. (1991), Minimizing weighted number of tardy jobs and weighted earliness-tardiness penalties about a common due date, *Computers and Operations Research*, **18**, 4, 379-389.
14. Lee, C.Y. and Kim, S.J. (1995), Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights, *Computers and Industrial Engineering*, **28**, 2, pp. 231-243.
15. Panwalkar, S.S., Smith, M.L. and Seidmann, A. (1982), Common due date assignment to minimize total penalty for the one machine scheduling problem, *Operations Research*, **30**: 391-399.
16. Smith, W.E. (1956), Various optimizers for single-stage production, *Naval Research Logistics Quarterly*, **3**:59-66.
17. Szwarc, W. (1989), Single-machine scheduling to minimize absolute deviation of completion times from a common due date, *Naval Research Logistics*, **36**:663-673.

# **Production Scheduling**

Abstracts

# LOWER AND UPPER BOUNDS FOR THE JOB-SHOP SCHEDULING PROBLEM WITH SEQUENCE-DEPENDENT SETUP TIMES

Christian Artigues (Corresponding author)

*Laboratoire d'Informatique, University of Avignon*

*CNRS FRE 2487, BP 1228, Avignon Cedex 9, France*

[christian.artigues@univ-avignon.fr](mailto:christian.artigues@univ-avignon.fr)

Fabrice Buscaylet

*Athelys*

*215 rue Jean-Jacques Rousseau, 92 136 Issy-Les-Moulineaux Cedex, France*

[fabrice.buscaylet@artelys.com](mailto:fabrice.buscaylet@artelys.com)

Dominique Feillet

*Laboratoire d'Informatique, University of Avignon*

*CNRS FRE 2487, BP 1228, Avignon Cedex 9, France*

[dominique.feillet@univ-avignon.fr](mailto:dominique.feillet@univ-avignon.fr)

**Abstract** We present several methods to minimize the makespan in a job-shop problem with sequence-dependent setup times. We first present active, semi-active and non-delay schedule generation scheme, from which priority-rule based sampling heuristics are derived. Focusing on the disjunctive graph representation of the problem, we discuss the loss of dominance properties of efficient neighborhoods for the standard job-shop problem. We define an alternative interchange neighborhood and we propose a simple tabu search method. We use the traveling salesman problem with time windows relaxation of the problem to derive new lower bounds that we embed in a dichotomizing exact tree search method. All together, the proposed methods significantly improve the best known results on all the 10 previously unsolved instances of Brucker and Thiele, 1996.

**Keywords:** job-shop scheduling, sequence dependent setup times, priority-rule based sampling, tabu search, TSPTW relaxation, branch-and-bound

## 1. Problem definition

In the job-shop scheduling problem with sequence dependent setup times, denoted  $J|s_{ij}|C_{\max}$ , a set of  $n$  jobs  $\mathcal{J} = \{J_1, \dots, J_n\}$  and a set of  $m$  machines  $\mathcal{M} = \{M_1, \dots, M_m\}$  are considered. Each job  $J_i \in \mathcal{J}$  is made of a chain of  $m$  operations  $O_{i1}, \dots, O_{im}$ . Each operation  $O_{ij}$  of  $J_i$  requires a single machine  $m_{ij} \in \mathcal{M}$ , all operations of the same job requiring distinct machines. Each operation  $O_{ij}$  has a non zero duration  $p_{ij}$ . Each operation  $O_{ij}$  must precede operation  $O_{i,j+1}$  for all  $i = 1, \dots, n$  and for all  $j = 1, \dots, m - 1$ . A sequence dependent setup time, denoted  $s_{ijk}$ , is defined for each couple of distinct jobs  $(J_i, J_j)$  and for each machine  $M_k$ . An initial setup time  $s_{0ik}$  is defined for each job  $J_i$  and for each machine  $M_k$ . It is assumed that the triangular inequality holds: for each machine  $M_k$  and for each triple of distinct jobs  $(J_i, J_j, J_x)$ , we have  $s_{ixk} \leq s_{ijk} + s_{jxk}$ . For each machine  $M_k$  and for each couple of distinct jobs  $(J_i, J_j)$ , we have  $s_{0jk} \leq s_{0ik} + s_{ijk}$ .

A non-negative starting time  $S_{ij}$  has to be assigned to each operation  $O_{ij}$  such that the makespan  $C_{\max} = \max_{i=1, \dots, n} S_{im} + p_{im}$  is minimized, subject to precedence constraints ( $S_{ij} + p_{ij} \leq S_{i,j+1}$ , for each  $i = 1, \dots, n$  and for each  $j = 1, \dots, m - 1$ ) and machine constraints ( $S_{ij} + p_{ij} + s_{ixk} \leq S_{xy}$  or  $S_{xy} + p_{xy} + s_{xik} \leq S_{ij}$  for each couple of distinct operations  $(O_{ij}, O_{xy})$  such that  $m_{ij} = m_{xy} = M_k$ ).

The job-shop scheduling problem with sequence dependent setup times is a NP-hard problem, as an extension of the standard job-shop scheduling problem, denoted  $J||C_{\max}$ , where all setup times are zero.

## 2. Literature review

Problem  $J|s_{ij}|C_{\max}$  received only little attention in the literature, albeit being a natural extension of the job-shop problem with many applications in manufacturing (see Allahverdi et al., 1999). Brucker and Thiele, 1996 proposed a branch-and-bound method for more general shop problems and generated a set of 15  $J|s_{ij}|C_{\max}$  instances, denoted by BT. Focacci et al., 2000 proposed a branch-and bound method in a constraint programming framework for another variant with alternative resources. Choi and Choi, 2002 proposed a local search method, also for an extension involving alternative operations. A simple tabu search heuristic has been tested on the BT instances by Artigues and Buscaylet, 2003. A branch-and-bound procedure is proposed by Artigues et al., 2004 for the  $J|s_{ij}|C_{\max}$  which improve the results obtained by Brucker and Thiele, 1996 and Focacci et al., 2000 on the BT instances thanks to the upper bounds obtained by a priority rule-based multipass heuristic proposed in Artigues et al., 2005. In the present paper we make a

synthesis of the methods described in Artigues et al., 2004, Artigues and Buscaylet, 2003 and Artigues et al., 2005. Furthermore, we obtain further significant improvements of our results on the BT instances by integrating tabu search and multi-pass sampling heuristics.

### 3. Multi-pass priority-rule based methods

We carry out an experimental study of several priority-rule based heuristics for the  $J|s_{ij}|C_{\max}$ , focusing especially on semi-active, active and non-delay schedule generation schemes (SGS). We show easily from a counterexample that a simple extension of the Giffler and Thompson, 1960 heuristic can be unable to compute the optimal solution. We then exhibit an alternative active SGS able to generate a set of dominant active schedules. We use the 3 proposed SGS as priority-rule based heuristics for the  $J|s_{ij}|C_{\max}$ . We test both single- and multi-pass methods. In the single pass method a unique priority rule is used to break the ties so as to obtain a single solution. Our conclusion, at least on the 15  $J|s_{ij}|C_{\max}$  instances and for the tested priority rules, is that the non-delay SGS obtains better results than the other ones. In the multi-pass framework, we select a priority rule and a random factor which gives the probability to select another operation than the one elected by the rule. Hence we make a sampling of the solutions reachable by a given SGS. As the number of iterations increases, it appears that the active and semi-active SGS with the MINSTART priority rule<sup>1</sup> are superior to the non-delay SGS. Since a single run of the active or semi-active SGS with the MINSTART rule (and another rule  $R$  to break the remaining ties) is equivalent to a run of the non-delay SGS with rule  $R$ , we conclude that the non-delay policy is a good one in the presence of sequence-dependent setup times and that it is profitable to search solutions in the neighborhood of non-delay schedules.

### 4. Tabu search

We study the extension of the disjunctive-graph based local search methods for the standard job-shop problem to the  $J|s_{ij}|C_{\max}$ . We restrict our attention to the block-based neighborhood proposed by Nowicki and Smutnicki, 1996, denoted as INTB (Interchange Near The Borderline). It aims at selecting a block of consecutive critical operations (located on a critical path of the disjunctive graph) and at allowing the interchange of 2 adjacent operations in a block, except for 2 internal operations. Furthermore, INTB is empty when there is a single block. Again, from a counterexample, we show that a local search method based on INTB may be unable to reach the optimal solution. Hence we pro-

pose an alternative neighborhood switching to a larger neighborhood (the interchange of 2 internal operations is allowed, even if there is only one block) when INTB becomes empty.

## 5. Lower bound computation and exact method

Removing the precedence constraints between the operations in the  $J|s_{ij}|C_{\max}$  gives  $m$  independent traveling salesman problems. We study such a relaxation, introducing operation time windows to obtain a strengthened relaxation made of  $m$  independent traveling salesman problems with time windows (TSPTW). The time windows are initialized using an upper bound of the makespan computed by the above-described heuristics. Then, the time windows are reduced by computing longest path computations in the precedence graph associated with basic resource constraint propagation techniques (namely disjunctive constraint propagation and edge-finding ignoring setup times, see Baptiste et al., 2001). The TSPTW relaxation is computed to obtain an initial lower bound. The optimal solution is then searched between the lower and upper bounds by a dichotomizing tree-search procedure. Based on a basic branching scheme, the procedure uses the decision variant of the TSPTW relaxation to prune the nodes of the search tree. Its originality lies in solving exactly the TSPTW relaxation by branch-and-bound, memorizing a limited number of found solutions in a dictionary. For each node (of the  $J|s_{ij}|C_{\max}$  tree), the dictionary is searched to check whether a previously computed TSPTW solution is still feasible. In the positive case, no solving of the relaxation is needed.

## 6. Computational experiments

We have tested the proposed methods on the 15  $J|s_{ij}|C_{\max}$  instances proposed by Brucker and Thiele, 1996. These instances have the particularity that each operation is associated with a setup type and the sequence-dependent setup time matrix is indexed by the setup types. In these instances however, all the operations of the same job have the same setup type<sup>2</sup>. The instances are made of 3 series of  $5 \times 10 \times 5$ ,  $5 \times 15 \times 5$  and  $5 \times 20 \times 10$  machines  $\times$  jobs  $\times$  setup types, respectively.

The priority-rule based sampling method has been coded in C++, whereas the tabu search method have been coded in CLAIRE. The exact method has been coded in C++ using ILOG Scheduler. The machine used is a Pentium 4 with 2.4 Ghz and 512 MB RAM. The makespan values obtained by the methods are given in table 6 for the 15 BT instances. The maximal time used by the sampling method with 10000 iterations (column H1) is always lower than 3 seconds. The maximal time used by



the tabu search method (column H2) does not exceed 41 seconds. The CPU times of the exact method (column H3 for the best upper bound and column LB for the best lower bound) are given in column CPU. When the CPU time is not indicated, the optimality could not be proven in a limit of 5 hours. The exact method is unable at the present time to solve the 5 largest problems. The results are compared with the previous best known solutions obtained by Brucker and Thiele, 1996 and Focacci et al., 2000 (column prev).

| inst | prev | H1   | H2   | H3   | LB   | CPU   |
|------|------|------|------|------|------|-------|
| 1    | 798  | 818  | 798  | 798  | 798  | 522   |
| 2    | 784  | 829  | 784  | 784  | 784  | 7.7   |
| 3    | 749  | 783  | 771  | 749  | 749  | 47.8  |
| 4    | 730  | 745  | 743  | 730  | 730  | 34.3  |
| 5    | 691  | 693  | 693  | 692  | 692  | 30.2  |
| 6    | 1056 | 1026 | 1026 | 1026 | 996  | -     |
| 7    | 1087 | 1039 | 1022 | 970  | 970  | 16650 |
| 8    | 1096 | 1019 | 994  | 1002 | 923  | -     |
| 9    | 1119 | 1060 | 1060 | 1060 | 1037 | -     |
| 10   | 1058 | 1036 | 1018 | 1018 | 1018 | 498   |
| 11   | 1658 | 1536 | 1509 | NA   | NA   | NA    |
| 12   | 1448 | 1319 | 1305 | NA   | NA   | NA    |
| 13   | 1549 | 1439 | 1439 | NA   | NA   | NA    |
| 14   | 1592 | 1492 | 1492 | NA   | NA   | NA    |
| 15   | 1744 | 1556 | 1556 | NA   | NA   | NA    |

Table 1. Results on the 15 BT  $J|s_{ij}|C_{\max}$  instances

The table shows that the exact method is able to solve 2 previously unsolved instances (7 and 10). However instance 7 is solved at the expense of large computational time and instance 10 is solved by initial upper and lower bound computations. The best known results are improved on all the 10 medium and large instances. The benefit of using the tabu search method appears on some instances, with an additional CPU time requirement. The medium quality results obtained by tabu search underline the need of testing more efficient neighborhoods. The quality of the initial TSPTW lower bound is good in comparison to the initial lower bounds computed for the branch-and-bound method of Brucker and Thiele, 1996 but rather costly in CPU time.

## 7. Conclusion and future work

We have studied the dominance properties of standard SGS and neighborhoods proposed so-far for the problem. With the heuristics issued

from this study, we have significantly improved the best known solutions on a set of  $J|s_{ij}|C_{\max}$  benchmark instances. The good upper bounds provided by these heuristics associated to a TSPTW based lower bound allowed us to close 2 medium size instances, with the help of a branch-and-bound method.

The proposed lower bounds could be further improved by a closer look of the TSPTW relaxations. We also intend to test new insertion-based neighborhoods.

## Notes

1. MINSTART: select the operation with the smallest starting time
2. In this case, the model is equivalent to the one presented in Section 1, which is not true in general.

## References

- Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27:219–239.
- Artigues, C., Belmokhtar, S., and Feillet, D. (2004). A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In J.C.-Régis and Rueher, M., editors, *1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Note in Computer Science 3011, pages 37–49. Springer.
- Artigues, C. and Buscaylet, F. (2003). A fast tabu search method for the job-shop problem with sequence-dependent setup times. In *Metaheuristic International Conference MIC'2003*.
- Artigues, C., Lopez, P., and Ayache, P. (2005). Schedule generation schemes and priority rules for the job-shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research*. to appear.
- Baptiste, P., Le Pape, C., and Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*. Kluwer Academic Publishers.
- Brucker, P. and Thiele, O. (1996). A branch and bound method for the general-shop problem with sequence-dependent setup times. *Operations Research Spektrum*, 18:145–161.
- Choi, I.-N. and Choi, D.-S. (2002). A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers and Industrial Engineering*, 42:43–58.
- Focacci, F., Laborie, P., and Nuijten, W. (2000). Solving scheduling problems with setup times and alternative resources. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 92–101.
- Giffler, B. and Thompson, G. (1960). Algorithms for solving production scheduling problems. *Operations Research*, 8(4):487–503.
- Nowicki, E. and Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813.

# PROACTIVE RESOURCE ALLOCATION HEURISTICS FOR ROBUST PROJECT SCHEDULING

Kristof Braeckmans, Erik Demeulemeester and Willy Herroelen

*Department of Applied Economics, K.U.Leuven, Belgium*

<first name>.<family name>@econ.kuleuven.ac.be

**Abstract** The well-known deterministic resource-constrained project scheduling problem (RCPSP) involves the determination of a *predictive schedule* (*baseline schedule or pre-schedule*) of the project activities that satisfies the finish-start precedence relations and the renewable resource constraints under the objective of minimizing the project duration. This pre-schedule serves as a baseline for the execution of the project. During execution, however, the project can be subject to several types of disruptions that may disturb the baseline schedule. Management must then rely on a *reactive scheduling* procedure for revising or reoptimizing the pre-schedule.

The objective of our research is to develop proactive heuristics for allocating resources to the activities of a given baseline schedule in order to maximize its stability. We develop several integer programming based models and heuristics such that the stability of the baseline schedule is maximized. We report on computational results on a set of benchmark problems.

**Keywords:** RCPSP, resource allocation, MIP

## 1. Stable resource allocation

We assume that the uncertainty during project execution resides in the activity durations that we assume to be beta distributed. Using average activity durations, a minimum duration baseline schedule is computed by the branch-and-bound algorithm of Demeulemeester and Herroelen, 1992. During project execution, these stochastic durations may take on a realization different from their expected values so that the initial schedule needs to be repaired.

The way in which resources are passed on between the various activities in the baseline schedule can be described by a resource flow network (Artigues and Roubellat, 2000). During project execution, these resource

flows affect the propagation of the schedule disruptions throughout the schedule. We assume that the resource flows are kept unchanged during project execution. Although this is a very strict policy, we can easily defend this assumption with the example of a scarce resource with a high set-up cost such as a crane.

Given a baseline schedule, our objective is to generate the resource flows such that the stability of the baseline schedule is maximized. As a measure for stability, we use  $\sum_{i \in N} w_i |E(S_i) - s_i|$ , the sum of the weighted absolute deviations of the start times  $s_i$  in the baseline schedule from the expected actual start times  $S_i$  realized during project execution.

## 2. Heuristics based on IP programming

To compute our feasible resource flow networks, we make use of linear and integer programming models. As the stability measure contains stochastic variables, which we cannot use as an input for an IP solver, we developed heuristic linear integer programming models with alternative linear objective functions.

For example, we can minimize the number of extra flow carrying arcs that are added to the model. Alternatively, the weighted sum of the total floats over all activities can be maximized.

## 3. Computational Experiment

We compare the results obtained by the IP models with the results obtained by previously developed resource flow network heuristics (Artigues and Roubellat, 2000 and Policella et al., 2004). All heuristics are encoded in C++ and we use ILOG CPLEX 8.1 to solve our IP models. We report on computational results obtained on a set of benchmark problems.

## References

- Artigues, C. and Roubellat, F. (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127:294–316.
- Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818.
- Policella, N., Oddi, A., S.F.Smith, and Cesta, A. (2004). Generating robust partial order schedules. In *Proceedings CP '04*.

# OPTIMAL PRODUCTION SCHEDULING FOR DAIRY INDUSTRIES

Philip Doganis, Haralambos Sarimveis, Alex Alexandridis and Panagiotis Patrinos

*National Technical University of Athens, School of Chemical Engineering,  
9, Heroon Polytechniou Str. Zografou Campus, Athens 15780, Greece*

Key words: Production Scheduling, MILP, Sales Forecasting, Dairy Products, Neural Networks, Genetic Algorithms

## ABSTRACT

During the few last years, the profile of the food industry has changed greatly. Local, small-sized plants have given their place to large, national or multinational-scale food processing units. Furthermore, the effort of the industry to respond to consumer demands for greater product variety has increased the number of production tasks significantly. For the above reasons, the production planning and scheduling process has become very complex and the available equipment is often used close to its maximum capacity.

The dairy industry in particular produces an increasing variety of products that differ in ‘internal’ features like fat content, flavor, the type of whey etc., or ‘external’, like the size of the container and the language of the label (for exportable goods). Moreover, there are restrictions regarding the suitability of a machine for a production task (i.e. some products can only be produced on some machines) as well as restrictions in the production sequence, which are directed by product properties. These constraints are translated into specific scheduling rules; for example, if skimmed milk is used for the production of zero-fat yogurt, it should be processed before low-fat or full-fat milk in order to achieve lower setup times and product losses due to changeovers. The presence of these restrictions, together with the need to maintain high quality standards in food production and the short shelf-life of dairy products, further complicate the scheduling process.

Production tasks are handled within the factory through production scheduling, which refers to the strategies of allocating equipment, utility and manpower resources over time to execute the processing tasks required to manufacture one or several products in time (Pinto & Grossmann, 1998). Obviously, the development of tools that can help the industry to increase its flexibility and the utilization of available resources emerges as a necessity.

Sales forecasting is another critical factor for the efficient operation of a food manufacturer. If an efficient sales forecasting system is available, it will be possible to prepare ahead of sudden surges or plunges in demand and reduce the amount of lost sales or excess production. That is important for the food industry in general, but especially for the dairy industry it is of great significance due to the short shelf-life of its products. Dairy products are particularly perishable and customers require that food is as fresh as possible. They are purchased several times within the week and their sales show intense fluctuations due to factors like price, promotions, changing consumer preferences or weather changes (Van der Vorst et al.,

1998). Because of these characteristics, when demand is less than expected, because of i.e. promotional activities of an antagonist, the unsold quantities will gradually become less likely to be preferred by consumers, as there are fresher products available. At the end of a brief life-cycle, unsold quantities have to be discarded, burdening the industry with extra cost for transportation and environmentally sound disposal. On the other hand, an underestimation of the demand creates a lost sales cost. Clearly, the accuracy of the sales forecast determines to a great extent the profitability of the company. However, companies are still in the development phase as far as forecasting methodologies are concerned. In a recent initiative 48% of companies were identified as poor at forecasting (Adenbajo & Mann, 2000).

In this work, a complete two-level framework for use in food and in particular dairy industries is proposed. The specific characteristics of the dairy industry have been taken into consideration, in terms of the behavior of food sales over time and the special requirements in the production phase. At the scheduling level, an MILP (Mixed Integer Linear Programming) model of the system was developed, using a continuous representation of time. The solution of the MILP problem provides a complete production schedule, which establishes in a precise manner the products to be produced on each machine, the respective quantities, and the starting and finishing time of each task. The model uses a detailed representation of events and their timing, in order to ensure that all the timing restrictions imposed by machine or feeding limitations, or even internal company requirements are respected. For example, in the case of a common feeding line, production must be coordinated across all machines. The proposed model allows incorporation of sequence-dependent transitions between products and calculates the respective setup time and cost. Furthermore, the produced schedule meets the production sequence limitations in terms of fat content, flavor, product whey and added ingredients.

At the forecasting level, a time-series model was developed based on the implementation of the Radial Basis Function (RBF) neural network architecture trained and the fuzzy means training algorithm (Sarimveis et al. 2002). Neural networks were preferred over linear models in order to capture the nonlinear elements of the dairy sales time series and were found to perform better compared to other methods that were tested. Furthermore, in order to optimize the performance of the forecasting model, a genetic algorithm approach was followed for the proper selection of input information. Forecasts are used as input to the scheduling methodology.

Application of the two-level framework on real data taken from the leading dairy product manufacturer in Greece illustrated the efficiency of the proposed method.

## ACKNOWLEDGEMENTS

Financial support by FAGE S.A. and the General Secretariat of Research and Technology in Greece under the PENED 2001 research program (01EΔ38) is gratefully acknowledged.

## REFERENCES

1. Adebajo, D., & Mann, R. (2000). Identifying problems in forecasting consumer demand in the fast moving consumer goods sector, *Benchmarking: An International Journal*, 7(3), 223-230.
2. Pinto, J.M. & Grossmann, I. E. (1998). Assignment and Sequencing Models for the Scheduling of Chemical Processes, *Annals of Operations Research*, 81, 433-466.
3. Sarimveis, H., Alexandridis, A., Tsekouras, G., & Bafas, G. (2002). A fast and efficient algorithm for training radial basis function neural networks based on a fuzzy partition of the input space, *Industrial and Engineering Chemistry Research*, 41, 751-759.
4. Van der Vorst, J. G. A. J., Beulens, A. J. M., De Wit, W. & Van Beek, P. (1998). Supply chain management in food chains: improving performance by reducing uncertainty, *International Transactions in Operational Research*, 5(6), 487-499.

## Approximations to Optimal $k$ -unit Cycles for Dual Gripper Robot Cells

H. Neil Geismar, Assistant Professor, Prairie View A&M University  
Milind Dawande, Associate Professor, University of Texas at Dallas  
Chelliah Sriskandarajah, Professor, University of Texas at Dallas

We consider the problem of scheduling operations in bufferless dual-gripper robot cells that produce identical parts. The objective is to find a cyclic sequence of robot moves that minimizes the long-run average time to produce a part or, equivalently, maximizes the throughput.

Our main result is a polynomial-time algorithm that generates a cyclic solution which is a 1.5-approximation to an optimal  $k$ -unit cycle ( $k \geq 1$ ) for a practically common subclass of cyclic solutions in cells with dual gripper robots, constant inter-machine travel-time, and the free-pickup criterion. Additionally, we show that this cyclic solution is asymptotically optimal as the number of machines goes to infinity. Over all cyclic solutions, we provide a polynomial-time algorithm which is a 2-approximation. We believe that our structural analysis is an important step toward resolving the longstanding open problem of finding an optimal  $k$ -unit cycle ( $k \geq 1$ ) in either a dual gripper or single gripper cell. As a by-product, we provide several managerial insights into the schedules that maximize productivity for dual gripper cells for any combination of processing requirements and physical parameters.

# The Unit-Operation-Stock Superstructure (UOSS) and the Quantity-Logic-Quality Paradigm (QLQP) for Production Scheduling in the Process Industries

Jeffrey Dean Kelly  
Honeywell Process Solutions, Toronto, Canada

July 18 to 21, 2005  
MISTA 2005 Conference, New York City.

MISTA – Multidisciplinary Conference on Scheduling Theory and Applications  
<http://www.mistaconference.org/>

## Abstract

It is well-known that production scheduling can be largely categorized into *open* and *closed-shops* (Graves (1981)). Open-shops are job-shops, flow-shops, machine-shops and project-shops and deal exclusively with the *assignment*, *sequencing* and *timing* decisions (Pinedo (1995)). Closed-shops on the other hand also deal with the same three decision-groups as open-shops but they also must decide on the *sizing* of lots, charges, batches and movements in order to properly program production especially with finite inventory capacity. Arguably open-shops are a subset of closed-shops. The purpose of this article is to overview a production model and its data which is comprehensive enough to schedule the breadth and depth of all plants found in the process industries. We achieve this by extending the state and resource-task networks (STN and RTN; Kondili *et. al.* (1993) and Pantelides (1994)) to explicitly include the aspects of limited routings between unit-operations and other unit-operations up and downstream in the material flow path as well as dealing directly with the complexity of charge/batch-splitting into movements. We call this novel production modeling approach the *unit-operation-stock superstructure* (UOSS) which is the cross-product of the physical (units) and the procedural (operations) models or substructures.

## Introduction

Although the unit-operation-stock superstructure (Kelly (2004b) and 2005)) defines the broad scope of the production network in terms the production's *connectivity* (and later *compatibility*) it does not describe the deep relationship between the many attributes and characteristics of the production's *capacity*. These capacity attributes detail the hydraulics, hold-up, operating procedures, processing rules, properties and conditions of the unit-operations involved in transforming feedstocks into productstocks via intermediatestocks and we represent this as the *quantity-logic-quality paradigm* (QLQP) (Kelly (2004a) and Kelly (2005)). Together, the UOSS and the QLQP define the production's *capability*. The production's capability is then used to solve in-series a mixed-integer linear program (MILP) representing the *production logistics* (quantity and logic) and a successive linear program (SLP) representing the *production quality* (quantity and quality). The idea of solving a logistics scheduling optimization first and then a quality scheduling optimization is to avoid the intractability of solving a MINLP though the breakdown is surprisingly natural and useful especially when debugging and diagnosing inconsistent and incapable problems. Moreover, this solution heuristic is structured similar to the structure of all types of heuristics in terms of performing a constructive or greedy search first and then a local or improvement search second and provides an effective approach to tackle the complexities of all process industry scheduling optimization problems. The sections to follow outline the UOSS and QLQP in the context of production.

Yet, before we proceed it seems prudent to discuss in more detail the definition of production. As with all complex systems, some form of anatomy or breakdown structure usually exists and production is no different. We believe that production is the tight integration and collaboration of *process*, *operations* and *maintenance*. Production is also the integration of other business functions such as procurement, environmental, safety, engineering, distribution and financial. The process element deals with the processing details of converting and transforming feedstocks to



productstocks known as the *chemistry* of the problem. The operations element deals with the operating details of running and monitoring the execution of making the productstocks known as the *physics* of the problem. And, the maintenance element deals with the maintaining details of sustaining and supporting both the processing and operating elements of production. And finally it is interesting to note that the *mathematics* of the problem is consigned to the decision-making of coordinating and controlling the process, operations and maintenance functions. Without the mathematics to proxy, abstract or surrogate the real world, it would be virtually impossible to model (think about) and manage the production in competitive markets and that is the focus of this article.

## Production Models (Model-Data)

As mentioned, a production model has essentially three components ultimately describing its capability i.e., its connectivity, capacity and compatibility. Production connectivity involves nodes and arcs. The nodes or vertices are the *logical* units or unit-operation pairs and the arcs are the directed edges between them which flows the various stocks. Figure 1 diagrams a small UOSS for a hypothetical plant with production objects we call perimeters, continuous-processes, batch-processes, pools, pipelines, parcels, ports, internal streams and external streams (see Kelly (2004b) for more details).

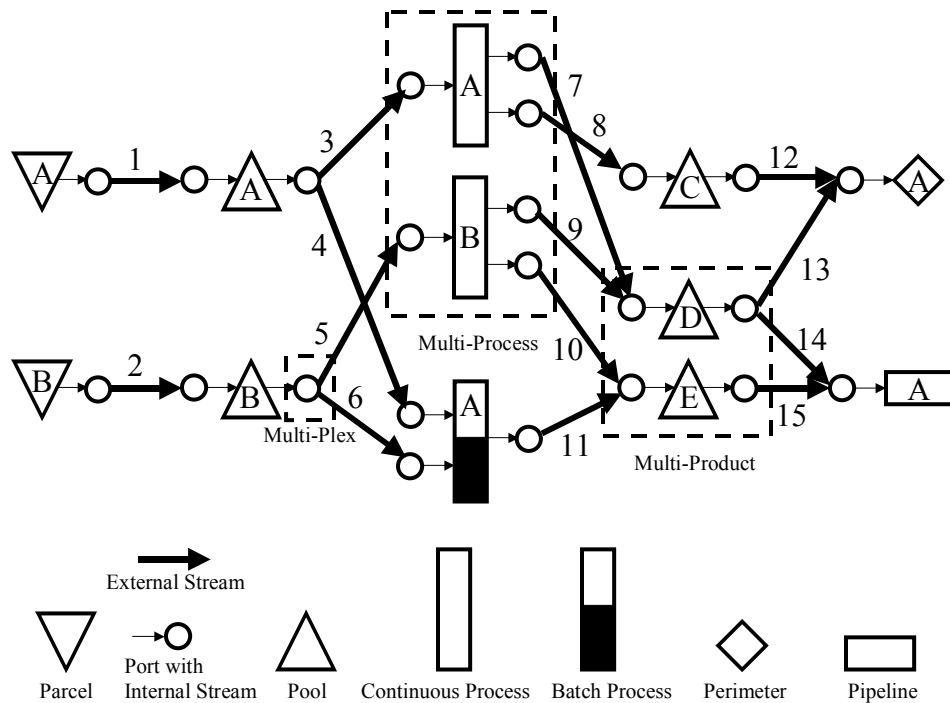


Figure 1. Production network or UOSS.

The units (renewable resources) are the physical or structural objects of production and are tangible (i.e., the plant). The operations are the procedural or behavioral objects and are intangible (i.e., the process) and are themselves broken down into *mode* (perimeters and processes), *material* (pools, pipelines, parcels, and ports) and *move* (port-to-port) operations as well as any sub-operations or operatives being required. The stocks (or “states” termed in the STN) are the materials or usually non-renewable resources such as the feedstocks, intermediatestocks and productstocks flowing through unit-operation-ports. Renewable resources can also flow within unit-operation-ports and are called *utilities* such as catalyst, steam, electrical, manpower and pecuniary resources. In essence, renewable resources are part of a *closed-system* given that they are re-circulated around the plant whereas non-renewable resources are part of an *open-system* given that they enter and leave the plant from the surroundings (i.e., sources and sinks) and do not necessarily reverse material-flow-path direction although recycles do exist and can make the

network cyclic with respect to both renewable and non-renewable resources. It should be noted that units that can perform only one operation are dedicated and units that can perform one or more operations (but not at the same time when making scheduling decisions) are called multi-purpose where Figure 1 shows three examples of this as indicated by the dotted-line boxes. In batch-processing, the context of production is primarily through the procedural or recipe view with the physical view being secondary. In continuous-processing the context is more along the physical or flowsheet view with the procedural being secondary. In the UOSS we describe both views together in a superstructure framework given that at the actual mathematics or decision-making level, both the physical and procedural aspects are necessary.

Production capacity involves all of the quantity, logic and quality attributes on objects listed above and can be found in Kelly and Mann (2003) and Kelly (2005b). Examples of logic attributes for example are semi-continuous flows, single-use restrictions (unary resource), up-time (for campaigns and cycles), down-time and switch-over-when-empty or full, etc. (Wolsey (1998)). The QLQP attributes translate into mathematical linear quantity variables (extensive), binary logic variables (intensive) and bilinear quality variables (intensive) as well as the equality and inequality quantity, logic and quality constraints satisfying in some form or another all of the quantity, logic and quality balances. The quantity variables are the lot, charge, batch and movement-sizings in the problem and are mainly involved in the quantity or material balances as well as semi-continuous logistics constraints. It is the explicit modeling of movement-sizes that differentiates the UOSS from the STN in light of the fact that a batches can be split into smaller quanta. There are matching logic variables for each quantity sizing variable and are associated with the unit-operation pairs and the unit-operation-port to downstream unit-operation-port connections or routes. These logic variables are non-linear or non-convex and are the “complicating” variables requiring us to use MILP with a branch-and-bound search. The quality variables are classed into compositions, properties and conditions. Compositions are the volume or weight fractions of a particular component in a stock or stream (i.e., a material within a material), properties can be a volume or weight-based empirical *chemical* property such as an octane, specific gravity or melting-index and conditions are the *physical* properties such as temperature and pressure but can represent other processing or operating conditions such as severity and conversion. It is the multi-linear (bi-linear, tri-linear, etc.) constraints of quantity times quality that require us to use non-linear programming solvers such as successive linear programming making the quality variables the so called complicating variables. Production compatibility is related to both connectivity and capacity in that it specifies the sequence-dependent and frequency-dependent (Kondili *et. al.* (1993)) repetitive/routine maintenance operations (i.e., non-productive operations) of cleaning, purging, washing and re-tooling of equipment before the next productive operation is allowed to take place. The compatibility of from-operations to to-operations within the same unit can also be used to model product-wheels that specify a rigid sequence or order for the transitions such as is found in paint-shops. Another important aspect of the production model is somewhat related to logic in that it involves the timing of events and defines the temporal decomposition of the scheduling horizon into relatively small time-buckets or periods. Big time-buckets are used in planning and can have one or more start-ups of the same unit-operation within the same time-period (Belvaux and Wolsey (1998)). However in scheduling, which uses small time-buckets, we can have at most one start-up of the same unit-operation within the same time-period. These time-periods are equally spaced meaning they all have the same time duration set exogenously. This type of time modeling is called discrete-time. Continuous-time models have recently become available and have endogenously determined time-period durations where the number of time-periods or time-points is set exogenously (Jia *et. al.* (2003)). We use discrete-time formulations for computational reasons though the UOSS can easily be extended to be formulated in continuous-time. All of the above production details related to the modeling of production is collected into what we call *model-data*. Model-data is relatively persistent or static and does not change from one scheduling execution to the next.

## Production Cycles (Cycle-Data)

A production cycle is our term to describe the fact that schedules are repeated usually once a day for primarily three reasons. The first is to mitigate against omnipresent uncertainty (Aytug *et. al.* (2005)). The second is to manage the spatial and temporal complexity and the third is to moderate the organizational structure and its various functions (i.e., for coordination and collaboration reasons). Cycles have primarily three elements and include both time-series (time-stamp, value) and transactional (start-time, end-time, value) data. The first are the production *commands* which provide the supply-orders for feedstocks, the demand-orders for productstocks and the repetitive, preventative and corrective maintenance-orders. There are also production-orders which as expected have two parts

and they are process-orders and operation-orders. Process-orders can be used to force or fix certain quantity and quality variables to be at certain limits some time in the future. Operation-orders can be used to fix logic variables to be either 0 (closed, off, inactive) or 1 (open, on, active). The second are the production *circumstances* or conditions which include such things as the opening inventories and compositions in pools (quantity and quality carry-over) and the amount of up-time left on a batch-process (logic carry-over). These circumstances establish the “state” of the production at the start of schedule (i.e., starting at time-period zero) by assessing the impact of past and present activities (i.e., negative time-periods). The third are the production *controls* which essentially provide the setpoints or targets as specified by a higher-level production (logistics and quality) planning optimization system. The targets will specify independent and dependent quantity, logic and quality variables as well as upper and lower bounds for these values in order to abate excessive excursions from setpoints (i.e., to keep them within a range or safe operating zone). These targets will be stewarded to by the logistics and quality scheduling optimizers similar to the notions of the 1-norm and 2-norm objective functions found in model predictive or advanced process control systems used to control many of the processes found in the process industries. All of the above production details related to the cycling of production is finally collected into what we call *cycle-data*. Cycle-data is dynamic and changes from one scheduling execution to the next (i.e., from day to day).

## Production Logistics (Solution-Data)

As alluded to previously, we solve a logistics scheduling optimization first as a constructive or greedy search. This optimization finds schedules or solutions that must satisfy the production logistics. To achieve this we use commercially available MILP codes such as CPLEX from ILOG and XPRESS from Dash Optimization formulated using constraints similar to those used in the fixed-charge flow network, facility location and lot-sizing/scheduling type problems (Nemhauser and Wolsey (1988), Williams (1993) and Wolsey (1998)). The global search variables are of course the logic or binary variables discussed above. Each unit-operation (modes and materials) and unit-operation-port to other unit-operation-ports (moves) are the independent binary variable degrees-of-freedom as well as the associated independent quantity sizing variables. There are also dependent quantity and logic variables worth mentioning. The dependent quantity variables are mostly the pool hold-up or inventories whereas the dependent logic variables are the start-ups, switch-overs-to-itself, switch-overs-to-other and shut-downs. The independent logic variables are what we call set-ups and it is for this reason that we relate the production logistics decisions as determining the *production set-up* contrasted with the production quality decisions called the *production settings* discussed below. Both the production set-up and settings determination problem is what we call *production programming* given that we are programming or circumscribing the production for the short, medium and long-term as accurately as possible by determining a suitable course of action. The logistical elements of the schedules generated by this function are stored into what we call *solution-data*.

The objective function used to describe the utility of the production logistics is to maximize profit, maximize performance and minimize penalties (Kelly and Mann (2003) and Kelly (2003b)). Profit is defined as productstock revenue minus feedstock costs minus holding costs on feedstocks, intermediatestocks and productstocks over the scheduling horizon. Performance is the minimization of the weighted absolute deviation of a pool quantity hold-up minus its end of schedule target and the minimization of all logic set-ups, start-ups, switch-overs and shut-downs. The idea behind minimizing set-ups etc. is to generate schedules which are as parsimonious in terms of effort. Penalties are dualized quantity and logic constraints manifested as artificial or elastic variables. All practical optimization problems must have some form of infeasibility breakers in order to handle the situation of ill-specified and over-aggressive problems.

Now to the major issue of intractability and the combinatorial nature of these non-convex MILP problems and how we manage to breed primal and integer-feasible solutions in reasonable time. The first step is to use sophisticated and continuously improving commercial MILP codes as mentioned which have powerful presolve/preprocessing capabilities to significantly reduce the number of linear and binary variables as well as the number of constraints including providing essential branch-and-cut technology. The second is to formulate the logistics sub-problems using as tight a formulation as possible by using the techniques available in Nemhauser and Wolsey (1988), Williams (1993) and Wolsey (1998) for example. And the third is to use greedy and local search type primal heuristics such as those found in Kelly (2002), Kelly (2003a) and Kelly and Mann (2004) which are general-purpose production logistics MILP heuristics designed to conservatively or aggressively reduce the number of binary variables. Other heuristic approaches known as meta-heuristics are available such as Ant Colony Optimization, Tabu Search, Simulated Annealing, Genetic Algorithms and Scatter Search but these are more

difficult to program given that the programmer needs to metaphorically translate the logistics details into their search paradigms.

The most successful primal heuristic approach that we have found so far is to adopt the *relax-and-fix* heuristic (Wolsey 1998) and to strategically “group” logic variables along the lines of the time, units and stock/operations dimensions (Elkamel *et. al.* (1997)). Specifically time-groups, unit-groups and stock-groups are prepared where a sequence of smaller MILP’s are run starting from the most important to the least important group fixing binary variables to the solution values found from the previous MILP run (Kelly and Mann (2004)). Another primal heuristic we call the *range-rounding* heuristic is also very effective at finding integer and primal-feasible solutions faster than running a branch-and-bound search alone is to compute two production extreme-points. These extreme-points are generated by maximizing and minimizing all quantity sizing variables in the objective function by solving two separate relaxed LP’s. If a logic variable is integral for the two solutions and has the same value (i.e., either zero or one) then we fix these logic variables to these values hence reducing, usually significantly, the number of binary search variables.

## Production Quality (Solution-Data)

The quality of production in our context relates to the quality variables and constraints defined as the composition, properties and conditions as well as all of the quantity and quantity times quality balances and bounds. Quality planning optimization is well established in the oil-refining and petrochemical industries (Bodington (1995) and Kelly (2004a)) using multi-period, multi-product and multi-plant models dating back to the early 1960’s. However, quality scheduling optimization is less well-known although it uses many of the same modeling and solving technologies including the details around the famous non-convex pooling problem (Greenberg (1995), Quesada and Grossmann (1995), Tawarmalani and Sahinidis (2002) and Poku *et. al.* (2004)). The objective function for the quality optimizer is similar to the logistics optimizer except that the performance and penalties terms are modeled using the quality variables instead of the logic variables; the profit term is the same for both problems. Quality performance is defined as the absolute weighted deviations from target and penalties are artificial variables attached to each quality lower and upper bound if an excursion from a bound exists.

All quality planning and scheduling optimization problems are in general multi-linear though other more general forms of non-linearities can be included. Multi-linear means that they involve linear, bi-linear, tri-linear, quad-linear, etc. type of polynomial structure. The simplest form of non-linearity is bi-linearity and in our formulation involves quantity times quality and quality times quality terms (Quesada and Grossmann (1995) and Kelly (2004c)). Quantity times quantity terms are not necessarily required although they can exist when complementarity relations are present to proxy, in a limited sense, logic constraints in the production quality optimization step (Gopal and Biegler (1999)). An example of this is to force only one flow out of a unit-operation-port in any given time-period when there are two or more destinations by imposing  $\text{flow1} * \text{flow2} \leq \alpha$  where  $\alpha$  is some small non-zero value. There are two important reasons for exploiting the structure of multi-linear problems also known as polynomial programs by tailoring the model and more importantly the optimizer to take advantage of this structure. The first is to use a successive linear programming (SLP, Palacios *et. al.* 1982)) solver as the optimizer instead of more generic and commercially available non-linear solvers such as CONOPT, GRG2, KNITRO, LOQO, MINOS or SNOPT. Because scheduling problems can involve many time-periods, SLP’s are well suited given that the sub-problem optimizer is an LP which has many years of expertise embedded in it for block-angular type matrices (Ponnambalam *et. al.* (1992)) and given the advancements being made around interior-point and primal/dual simplex codes. SLP is also the solver of choice for quality planning optimization and thus SLP for quality scheduling optimization can leverage off of this large domain experience. The presolving capability developed for LP’s can be directly applied to SLP in addition to the fact that MILP presolving notions required for *bi-nary* (logic) variables can be partially extrapolated to bi-linear (quality) variables (Lodwick (1992)). The second is to make use of the fact that multi-linear problems can be “cascaded” from one major iteration to the next. This is a concept utilized in the SLP code from Dash Optimization called XPRESS-SLP (Main (1993)). The basic idea is to re-compute the dependent quantity and quality variables from the independent quantity and quality variables. After an LP has converged in the SLP executive, the LP has generated quantity and quality “linearized-estimates” for the multi-linear dependent variables using the first-order Taylor-series expansion of the multi-linear terms and constraints. Obviously, these estimates can be of relatively poor quality and can be improved similar to the iterative improvement techniques found in Axelsson (1996) for numerical matrix inversion. By running cascaded LP’s in-series to properly estimate the dependent quantity and quality variables from the independent quantity and quality

solution values, better dependent variable values can be computed. The number of cascaded LP's in-series is a function of the degree of multi-linearity. If the quality problem is bi-linear then we require one cascaded LP after each SLP LP run, if tri-linear we require two cascaded LP's and so on. This idea is also used in Kelly and Mann (2003) to compute reasonable starting values for all qualities based the exogenous quantity values set by the logistics scheduling optimizer. That is, when running logistics optimization first we have reasonable estimates of the independent quantity variables from which to derive excellent initial values for the SLP.

## Summary

It should be emphasized that all of the concepts discussed above can be found in a commercial production scheduling application available by Honeywell Process Solutions. The next step in optimization would be to perform simultaneous logistics and quality optimization using MISLP. Unfortunately, for industrial-size problems of any practical interest makes this approach intractable for the near future in spite of the tremendous advancements in computer hardware and computer and mathematical programming algorithms. And finally, even if useable MISLP existed, when actual or apparent infeasibilities and/or highly resource constrained scheduling problems exist, scheduling logistics first then quality second can provide a powerful analysis tool in that a structured decomposition of the problem is afforded.

## References

- Axelsson, O., *Iterative Solution Methods*, Cambridge University Press, New York, (1996).
- Aytug, H., Lawley, M.A., McKay, K., Mohan, S. and Uzsoy, R., "Executing production schedules in the face of uncertainties: a review and some future directions", *European Journal of Operational Research*, **161**, 86-110, (2005).
- Belvaux, G. and Wolsey, L.A., "Lot-sizing problems: modeling issues and a specialized branch-and-cut system bcprod", *CORE Discussion Paper*, **DP9848**, Universite Catholique de Louvain, February, (1998).
- Bodington, C.E., (editor), *Planning, scheduling and control integration in the process industries*, McGraw-Hill Inc., San Francisco, (1995).
- Elkamel, A., Zentner, M., Pekny, J.F. and Reklaitis, G.V., "A decomposition heuristic for scheduling the general batch chemical plant", *Engineering Optimization*, **28**, 299-330, (1997).
- Gopal, V. and Biegler, L.T., "Smoothing methods for the treatment of complementarity conditions and nested discontinuities in chemical process engineering", *AIChE Journal*, **45**, 7, 1535-1547, (1999).
- Graves, S. C., "A review of production scheduling", *Operations Research*, **29**, 4, 646-675, (1981).
- Greenberg, H.J., "Analyzing the pooling problem", *ORSA Journal on Computing*, **7**, 2, 206-217, (1995).
- Jia, Z., Ierapetritou, M.G. and Kelly, J.D., "Refinery short-term scheduling using continuous-time formulation: crude-oil operations", *Industrial and Engineering Chemistry Research*, **42**, 3085-3097, (2003).
- Kelly, J.D., "Chronological decomposition heuristic for scheduling: a divide & conquer method", *AIChE Journal*, **48**, 2995-2999, (2002).
- Kelly, J.D. and Mann, J.L., "Crude-oil blend scheduling optimization: an application with multi-million dollar benefits – parts I and II", *Hydrocarbon Processing*, June/July, 47-53/72-79, (2003).
- Kelly, J.D., "Smooth-and-dive accelerator: a pre-milp primal heuristic applied to scheduling", *Computers & Chemical Engineering*, **27**, 827-832, (2003a).
- Kelly, J.D., "Next generation refinery scheduling technology", *NPRA Plant Automation and Decision Support Conference*, San Antonio, Texas, September (2003b).
- Kelly, J.D., "Formulating production planning models", *Chemical Engineering Progress*, January, 43-50, (2004a).
- Kelly, J.D., "Production modeling for multimodal operations", *Chemical Engineering Progress*, February, 44-46, (2004b).
- Kelly, J.D., "Formulating large-scale quantity-quality bilinear data reconciliation problems", *Computers & Chemical Engineering*, **28**, 3, 357-366, (2004c).
- Kelly, J.D. and Mann, J.L., "Flowsheet decomposition heuristic for scheduling: a relax-and-fix method", *Computers & Chemical Engineering*, **28**, 2193-2200, (2004).
- Kelly, J.D., "Modeling production-chain information", *Chemical Engineering Progress*, February, (2005a).
- Kelly, J.D., "The logistics of blending: the missing-link in blend scheduling optimization", *Hydrocarbon Processing*, September, (2005b).

Kondili, E., Pantelides, C.C. and Sargent, R.W.H., "A general algorithm for short-term scheduling of batch operations – I milp formulation", *Computers & Chemical Engineering*, **17**, 211-227, (1993).

Lodwick, W. A., "Preprocessing nonlinear functional constraints with application to the pooling problem", *ORSA Journal on Computing*, **4**, 2, 119-131, (1992).

Main, R.A., "Large recursion models: practical aspects of recursion techniques", in *Optimization in industry*, Ciriani, T.A. and Leachman, R.C., (editors), John Wiley & Sons Ltd., New York, (1993).

Nemhauser, G.L. and Wolsey, L.A., *Integer and combinatorial optimization*, John Wiley & Sons, New York, (1988).

Palacios-Gomez, F., Ladson, L., and Enquist, M., "Nonlinear optimization by successive linear programming", *Management Science*, **28**, 10, 1106, 1120, (1982).

Pantelides, C.C., "Unified framework for optimal process planning and scheduling", *Proc. Second Conf. on Foundations of Computer Aided Operations*, Rippin, D.W.T. and Hale, J. (editors), CACHE Publications, 235-274, (1994).

Pinedo, M., *Scheduling: theory, algorithms and systems*, Prentice Hall, New Jersey, (1995).

Poku, M.Y.B., Biegler, L.T. and Kelly, J.D., "Nonlinear optimization with many degrees of freedom in process engineering", *Industrial Engineering Chemistry Research*, **43**, 6803-6812, (2004).

Ponnambalam, K., Vannelli, A. and Woo, S., "An interior point method implementation for solving large planning problems in the oil refinery industry", *Canadian Journal of Chemical Engineering*, **70**, 368-374, (1992).

Quesada, I., and Grossmann, I.E., "Global optimization of bilinear process networks with multicomponent flows", *Computers & Chemical Engineering*, **19**, 2, 1219-1242, (1995).

Tawarmalani, M., and Sahinidis, N.V., *Convexification and global optimization in continuous and mixed-integer nonlinear programming*, Kluwer Academic Publishers, London, (2002).

Williams, H.P., *Model building in mathematical programming*, 3rd Edition, John Wiley & Sons, (1993).

Wolsey, L.A., *Integer Programming*, John Wiley & Sons, New York, (1998).

# **FORMING AND SCHEDULING JOBS WITH ORDER WEIGHTS IN SEMICONDUCTOR MANUFACTURING**

Fehmi Tanrisever and Erhan Kutanoglu

*The University of Texas at Austin*

**Abstract:** We study a new scheduling problem motivated by the challenges observed in the newest semiconductor manufacturing wafer fabrication facilities. As wafers are larger and heavier in these wafer fabs, it is becoming more common to use specialized material handling containers that carry multiple orders coming from different customers and to schedule the containers as jobs in the fab. Noting that orders may have varying levels of importance, we consider the system performance as a function of the weighted completion times of orders. The performance ultimately depends on both (1) how the orders are assigned to such containers (“job formation”), and (2) how the containers are scheduled in the fab (“job scheduling”). The overall problem is to find the best way to form and schedule the jobs subject to complicating constraints, including the restrictions on the number of containers that can be used at one time and on the number of wafers the containers can carry. In this study, we extend our earlier results provided for the unweighted completion time case, and show that some of the important properties do not extend to the weighted problem. In particular, solving the weighted problem is significantly harder than solving the unweighted problem. To solve the weighted problem, we introduce a heuristic method, based on the ideas presented in an earlier study. We report our preliminary computational results, which show that the heuristic provides near-optimal solutions.

**Key words:** Multiple Orders per Job, Semiconductor Manufacturing, Job Formation and Scheduling, Batching, Weighted Completion Time.

# A TABU-SEARCH BASED REACTIVE PROCEDURE FOR THE RCPSP WITH STOCHASTIC RESOURCE AVAILABILITIES

Olivier Lambrechts, Erik Demeulemeester and Willy Herroelen

*Katholieke Universiteit Leuven*

*Department of Applied Economics*

<first name>.<last name>@econ.kuleuven.ac.be

**Abstract** We consider an extension of the RCPSP in which uncertainty is introduced by means of stochastic resource breakdowns. These breakdowns can cause schedule infeasibilities that can be resolved by postponing one or more activities. In order to repair the disrupted schedule we developed a reactive approach based on tabu search that tries to minimize the weighted deviation from the original baseline schedule.

**Keywords:** project scheduling, resource breakdowns, tabu search

## Introduction

A project consists of a set of precedence related activities that have to be executed within a given timeframe. Each activity has a fixed duration and requires a fixed amount of a given resource type per period. The planned execution of such a project is described by a baseline schedule. This baseline schedule indicates when each activity starts and has to be precedence as well as resource feasible. Resource feasibility implies that there exists no time period in which the cumulative resource requirements of all active activities exceed the resource availability. We will consider the case where these resource availabilities can vary due to unforeseen resource breakdowns. Such breakdowns can lead to resource infeasibilities that can be solved by postponing one or more activities. Starting from the current schedule, our goal is to find a new feasible schedule that deviates as little as possible from the original baseline schedule. The objective we try to minimize is the project instability, defined as the sum of the weighted deviations between planned and observed activity starting times. Avoiding these deviations is important because often work is subcontracted or executed by resources that do



not belong exclusively to the current project. A change in the starting times of subcontracted or similar activities could lead to global infeasibilities (resource conflicts on the multi-project level) or to penalties in the form of higher subcontracting costs.

## 1. Reactive Policies

In order to reschedule the unfinished activities we developed a tabu search heuristic (Glover and Laguna, 1993). Our algorithm is based on a tabu search procedure for solving the deterministic RCPSP (Pinson et al., 1994). Solutions are represented by means of the priority list representation scheme and can be decoded again into a feasible schedule by using the serial scheduling scheme. The procedure starts from the priority list representation of the disrupted schedule. The solution neighbourhood is defined as those solutions that can be obtained by swapping two activities in the priority list of the current solution. In each step the procedure evaluates each solution belonging to this neighbourhood and selects the non-tabu move with the best value for the objective function. After a given number of iterations the procedure terminates with the currently best solution to the rescheduling problem. In order to diversify the search and to avoid local optima, frequency based penalties are included.

## 2. Computational Results

We test the performance of our tabu search algorithm by means of simulation for the 30-activity networks of the PSPLIB set of problem instances (Kolisch and Sprecher, 1997) using various proactive strategies for setting the baseline schedule. The approach is then compared to some simple reactive policies based on priority rules. Preliminary results show that the tabu search procedure strongly outperforms the best priority rule policy and this as well for protected as for unprotected baseline schedules.

## References

- Glover, F. and Laguna, M. (1993). *Tabu Search*. Blackwell Scientific, Oxford. In C. Reeves (Editor): *Modern Heuristic Techniques for Combinatorial Problems*.
- Kolisch, R. and Sprecher, A. (1997). PSPLIB - a project scheduling library. *European Journal of Operational Research*, 96:205–216.
- Pinson, E., Prins, C., and Rullier, F. (1994). Using tabu search for solving the resource-constrained project scheduling problem. In *Fourth International Workshop on Project Management and Scheduling*, pages 102–106. Euro Working Group on Project Management and Scheduling.

# A SURVEY ON MINIMIZING MAKESPAN FOR THE PREEMPTIVE JOB SHOP WITH TWO MACHINES

Tracy Kimbrel

*IBM T.J. Watson Research Center*  
kimbrel@us.ibm.com

We survey results for the job shop scheduling problem  $J2|pmtn|C_{\max}$  for the case of two machines with preemption. With respect to polynomial time approximation algorithms, the problem is essentially equivalent to the case in which there are two machines and each operation has unit length. This assumption can be achieved in polynomial time with only  $\epsilon$  loss by scaling and rounding the inputs to give polynomial-valued operation lengths. Then since preemption is allowed we may take an operation to be composed of multiple unit-length operations (on the same machine). (The assumption can be removed with no loss for all but the last algorithm presented but will be used throughout to simplify descriptions.)

The problem then reduces to a constrained matching problem as follows. We consider each job to be a directed chain of precedence arcs between unit length operations. We seek a matching (set of undirected arcs) such that there are no directed cycles in the mixed graph containing both the precedence and matching arcs. Each matching arc is between an operation on machine  $M_1$  and an operation on machine  $M_2$  which will be performed concurrently. It is straightforward to derive a valid schedule from this graph.

- 1 Hefetz and Adiri give a polynomial-time exact solution in the case that each job contains a sequence of unit-time operations that always alternate between  $M_1$  and  $M_2$ . [Hefetz and Adiri, 1982]
- 2 Jansen, Solis-Oba, and Sviridenko give a polynomial-time approximation scheme (PTAS) in case there are a constant number of operations per job (without any transformation to unit-time operations) both for the preemptive and nonpreemptive problems. This algorithm applies to the case of any constant number of machines, not just two. [Jansen et al., 2003]
- 3 Sevastianov and Woeginger give a linear-time  $\frac{3}{2}$ -approximation algorithm for minimizing the makespan. The algorithm divides the jobs into two sets  $S$  and  $U$ . It matches operations as follows. Work from  $S$  on  $M_1$  is matched as much as possible to work from  $U$  on  $M_2$ . No attempt is made to match operations in  $S$  on  $M_2$  to operations in  $U$  on  $M_1$ . Because of this, it is quite simple to find a matching that does not induce any cycles. The authors show that for some instances, an appropriate choice of  $S$  and  $U$  will yield makespan at most  $3L/2$  where  $L$  is the maximum of the loads on machines  $M_1$  and  $M_2$ . For other instances, the jobs must be divided into three sets, and two iterations of the matching process are needed. Again, for an appropriate partitioning, the makespan is shown to be at most  $3L/2$ . [Sevastianov and Woeginger, 1998]
- 4 Kimbrel and Saia match the approximation ratio of  $3/2$  in the randomized online setting. The algorithm requires only a single random bit and thus is trivially derandomized to also give a linear-time offline algorithm. [Kimbrel and Saia, 2000]

This algorithm starts with an arbitrary permutation  $\pi$  of the  $n$  jobs. It chooses randomly between two priority schemes. Under one scheme, priority for machine  $M_1$  is given to jobs according to  $\pi$  and priority for machine  $M_2$  is given according to  $\pi^r$ . Under the second scheme, these priority orderings are reversed.

A schedule is constructed by placing each job in one of two queues, corresponding to the machine required by its first operation, ordered by priority. The online algorithm repeatedly matches the first operations of the jobs at the heads of the two queues and removes those operations from the jobs. Each of these two jobs is reinserted into the opposite queue at the proper position according to the priority ordering if needed (i.e., the next operation is on the other machine). If the operation removed was the last, the job is simply removed from the queue. If all jobs are in the same queue, no match is possible and the first operation in the job at the head is simply removed. The sum of lengths of the schedules given by the two priority schemes is at most 3 times the optimum.

- 5 Anderson *et al.* give an algorithm with a tighter approximation guarantee of  $L + \ell/2$  in terms of the maximum machine load  $L$  and the maximum job length  $\ell$ , but this is still  $3/2$  in the worst case. The algorithm matches work on both machines from each job (except two), and interleaves the work from different jobs in a careful way. [Anderson et al., 2001]

Without loss of generality assume that the load on machine  $M_1$  is  $L$ . The algorithm chooses any permutation  $\pi$  such that all jobs with more operations on  $M_1$  than on  $M_2$  precede all other jobs. It considers only matching arcs between operations  $o_1$  and  $o_2$  in jobs  $j_1$  and  $j_2$  such that  $o_1$  is on  $M_1$ ,  $o_2$  is on  $M_2$ , and  $j_1 < j_2$ . Operations on machine  $M_2$  are ordered as follows: the decreasing priority order consists of the operations in  $j_1$  in precedence order, followed by operations in  $j_2$ , and so on. Similarly, operations on machine  $M_1$  are ordered as follows: the decreasing priority order consists of the operations in  $j_n$  in precedence order where  $n$  is the number of jobs, followed by operations in  $j_{n-1}$ , and so on.

The operations on machine  $M_2$  are processed in decreasing priority order. When looking for matches for operations in job  $j$ , all operations on  $M_1$  in jobs up to and including job  $j - 1$  have been released into a pool of operations available for matching. These operations are consumed in decreasing priority order.

Note that this algorithm provides an approximation ratio better than  $3/2$  except in extreme instances in which a single job contains about half the total work or more.

- 6 The approximation results above for this problem use as a lower bound the maximum of  $L$  and  $\ell$ . Sevastianov and Woeginger note that any approximation algorithm with ratio better than  $3/2$  for the two machine case would require a new, non-trivial lower bound on the optimal makespan. Bansal *et al.* give such a result. Their algorithm has an approximation ratio of less than 1.45. [Bansal et al., 2005]

In case the largest job has significantly less than half the total work, the algorithm of Anderson et al. can be used. If the largest job has significantly more than half the work, the length  $\ell$  of this job gives a good enough lower bound on the optimal schedule length so that any reasonable schedule (that does not simultaneously idle both machines) has makespan less than  $3/2$  times the optimal.

If there is a “big” job  $B$  containing about half the total work, the algorithm of Bansal et al. attempts to maximize the number of operations of other (“small”) jobs that are matched to operations of  $B$ . No attempt is made to match any two operations both belonging to jobs other than  $B$ . This special case of the matching problem is solved via linear programming relaxation of an integer programming formulation and rounding so that the number of operations performed concurrently with those in  $B$  is at least a  $(1 - 1/e)$  fraction of the

most possible. Even if the optimal schedule matches all these and all remaining operations among jobs other than  $B$ , the lower bound so obtained is shown to be enough to guarantee an approximation ratio less than  $3/2$ .

The LP can be informally described as follows: Consider an operation  $i$  of a small job  $j$  and imagine traversing the operations  $k$  of  $B$  in precedence order. At step  $k$  match a fraction  $x_{i,j,k}$  of  $i$  to  $k$ , and then discard (i.e., leave unmatched) a fraction  $y_{i,j,k}$  of  $i$  before moving on to stage  $k + 1$ . These matchings must obey precedence constraints in a fractional sense; that is,

$$\sum_{t=0}^{k-1} (x_{i-1,j,t} + y_{i-1,j,t}) - \sum_{t=0}^{k-1} (x_{i,j,t} + y_{i,j,t}) - x_{i,j,k} \geq 0.$$

The solution is rounded as follows:

Let  $0 \leq s_{i,j,k} \leq 1$  denote the extent to which operation  $i$  of job  $j$  has been matched to operations  $1, \dots, k - 1$  of job  $B$  or discarded during the first  $k - 1$  steps, i.e.,  $s_{i,j,k} = \sum_{t=1}^{k-1} (x_{i,j,t} + y_{i,j,t})$ . Let  $v_{i,j,k} = s_{i,j,k} + x_{i,j,k}$ ; thus  $v_{i,j,k} - s_{i,j,k}$  is exactly the extent to which  $i$  is matched to  $k$ . Also note that  $y_{i,j,k} = s_{i,j,k+1} - v_{i,j,k}$ . Note that the fractional precedence constraints imply that  $s_{i-1,j,k} \geq v_{i,j,k}$ .

- (a) For each job  $j$  choose  $u_j \in [0, 1]$  uniformly at random.
- (b) For each operation  $i$  in job  $j$  assign  $i$  to operation  $k$  of  $B$  if and only if  $s_{i,j,k} \leq u_j < v_{i,j,k}$ .
- (c) Let  $N(k)$  denote the number of operations assigned to  $k$ .  $N(k)$  is a random variable. If  $N(k) = 0$ ,  $k$  is not matched to any operation. If  $N(k) = 1$ , match  $k$  to the unique  $i$  assigned to it. If  $N(k) \geq 2$ , arbitrarily match  $k$  to one of the operations assigned to it, and discard the remaining  $N(k) - 1$  operations (never to be matched again).

Combining these different cases gives a worst case approximation ratio of about 1.442 for the general case.

## References

- Anderson, E., Jayram, T., and Kimbrel, T. (2001). Tighter bounds on preemptive job shop scheduling with two machines. *Computing*, 67:83–90.
- Bansal, N., Kimbrel, T., and Sviridenko, M. (2005). Job shop scheduling with unit processing times. In *Proceedings of the ACM/SIAM Symposium on Discrete Algorithms*, Vancouver, British Columbia, Canada.
- Hefetz, N. and Adiri, I. (1982). An efficient optimal algorithm for the two-machine, unit-time job-shop schedule-length problem. *Mathematics of Operations Research*, 7:354–360.
- Jansen, K., Solis-Oba, R., and Sviridenko, M. (2003). Makespan minimization in job shops: a linear time approximation scheme. *SIAM J. Discrete Math.*, 16:288–300.
- Kimbrel, T. and Saia, J. (2000). On-line and off-line preemptive two-machine job shop scheduling. *Journal of Scheduling*, 3:355–364.
- Sevastianov, S. and Woeginger, G. (1998). Makespan minimization in preemptive two machine job shops. *Computing*, 60:73–79.

# Single Machine Product Scheduling with Outsourcing Allowed

C. S. Sung<sup>a</sup>, Ik Sun Lee<sup>a</sup>, S.H. Yoon<sup>b</sup> and J.H. Juhn<sup>c</sup>

<sup>a</sup> Dept of Industrial Engineering, KAIST, Yuseong-gu, Daejeon 305-701, Korea

<sup>b</sup> School of IT and Engineering, Jeonju University, Korea

<sup>c</sup> Dept of Comp Public Admin, Chongju National College of Science & Technology, Korea

## Abstract

This paper considers various single-machine product scheduling problems with outsourcing allowed, where each product can be either produced by the machine or outsourced at market. Outsourcing has been becoming increasingly popular as a way to improve productivity in various companies, especially, including electronics industries and motor industries. The objective is to find the optimal product sequencing (scheduling) with outsourcing (if needed) allowed, which minimizes sum of the associated outsourcing cost and scheduling cost. The scheduling cost is represented by each of the scheduling measures including makespan, (weighted) sum of completion times, maximum lateness, and sum of lateness.

In the analysis, the proposed problem of minimizing sum of the associated scheduling cost and the outsourcing cost, called *unconstrained problem*, and that of minimizing the scheduling cost subject to outsourcing budget, called *constrained problems*, are analyzed for their complexities. Then, some associated solution properties are characterized to derive DP algorithms.

# AN EVALUATION OF EXACT AND SUBOPTIMAL PROCEDURES FOR CONSTRUCTING STABLE PROJECT SCHEDULES

Stijn Van de Vonder, Erik Demeulemeester and Willy Herroelen

*Department of Applied Economics,*

*K.U.Leuven, Belgium*

<firstname>.<last name>@econ.kuleuven.be

**Abstract** Solution robust project scheduling is a growing research field aiming at constructing proactive baseline schedules that anticipate multiple disruptions during project executions. When activity durations are assumed to be stochastic, including time buffers (gaps) in the baseline schedule to absorb disruptions is a proven method to improve the stability of the baseline schedule. The objective of this paper is to introduce and compare exact and heuristic algorithms for allocating buffers into a schedule.

**Keywords:** Project scheduling, uncertainty, stability, buffers

## Introduction

In this paper, we construct stable baseline schedules for projects with stochastic activity durations. Maximizing stability in this context means minimizing the expected weighted deviation between the preschedule and the realized schedule during project execution (Leus, 2003). Recent research (Van de Vonder et al., 2004) showed that this objective function is an interesting alternative for the traditional  $\min C_{max}$  for realistic project settings. We refer to Leus and Herroelen, 2005 for  $NP$ -hardness proofs of different stable scheduling problems

The literature concerning stable buffer allocation is almost void. We propose multiple heuristic algorithms. The simple RFDFH heuristic proposed in Van de Vonder et al., 2004 is used as a benchmark method. An exact partial enumeration procedure for smaller projects is also introduced. We provide an extensive computational analysis of all considered algorithms.

## 1. Algorithms

A qualitative buffer allocation procedure to include safety in a project schedule should mainly consider two important project characteristics. First, the size of the buffer protecting an activity  $i$  should be positively correlated with the marginal cost  $w_i$  of delaying that activity. Second, also the amount of variability in the durations of all activities that precede activity  $i$  in the schedule is relevant information to decide upon the size of the required buffer in front of  $i$ .

Our benchmark heuristic RFDDFF is only concerned with the relative proportions of all  $w_i$ 's. The newly proposed single-pass algorithms will also include available information on the activity duration variances.

We also propose a steepest descent and a tabu search algorithm to solve the introduced problem. Last, an exact procedure for allocating time buffers into a schedule is presented. The procedure is based on a partial enumeration of all possible solutions. The search tree is pruned by some dominance rules.

## 2. Experimental set-up and results

All algorithms have been coded in C++. Extensive simulation on a large set of project networks is used to evaluate the objective function. The impact of the activity durations variance structure on the results is also discussed in detail.

We analyze the computational efficiency of the exact procedure and the effectiveness of the dominance rules.

## References

- Leus, R. (2003). *The generation of stable project plans*. PhD thesis, Department of applied economics, Katholieke Universiteit Leuven, Belgium.
- Leus, R. and Herroelen, W. (2005). The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33:151–156.
- Van de Vonder, S., Demeulemeester, E., Herroelen, W., and Leus, R. (2004). The trade-off between stability and makespan in resource-constrained project scheduling. Technical Report 0423, Department of Applied economics, Katholieke Universiteit Leuven, Belgium.

# **Real-time Scheduling**

Full Papers



# Periodic real-time scheduling: from latency-based model to deadline-based model

Liliana Cucu\*, Yves Sorel

INRIA Rocquencourt,  
BP 105 - 78153 Le Chesnay Cedex, France  
*liliana.cucu@inria.fr, yves.sorel@inria.fr*

## Abstract

*We evoke our non-preemptive model of real-time systems with strict periodicity constraints, which is a latency-based model. We also evoke the classical non-preemptive model of real-time systems with release times and periodicity constraints, which is a deadline-based model. We give an algorithm which transforms an instance of the scheduling problem using the classical deadline-based model in an instance of the scheduling problem using our latency-based model. We show that the two problems are equivalent, thus the scheduling algorithm given for the problem using our model solves also the problem using the classical model. We present a modified version of this scheduling algorithm, which takes into account the deadlines instead latencies.*

**Keywords:** scheduling, real-time, latency, deadline, periodicity.

## 1 Introduction and existing results

Real-time systems are first of all reactive systems, that is, to each event arriving in the system through sensors corresponds an output event produced through actuators to the environment in reaction to this input event. This behavior is, usually, described using periodic systems of operations. In the classical deadline-based model given in [1], an operation becomes available at its release time which is repeated periodically and its start time is greater or equal to its release time. The operation must be scheduled before its deadline which is relative to the release time. The formal definition of this model is given in section 2. From now on the classical model is called deadline-based model.

When the real-time applications are based on signal, image and control processing algorithms [2], strict periodicity constraints are defined. In this case, an operation is always available and the notion of release time has no meaning. The start times are periodic and there is no deadline.

In our model given in [3] beside the strict periodicity constraints, two other types of constraints may be imposed on these systems. Firstly, because some operations can produce data consumed by other operation(s) as it is the case when the application functionalities are specified with a tool like Simulink, precedence constraints are imposed onto these operations. Secondly, for example in the case of brake control in a car, when the driver presses the brake pedal, the wheels must stop before a fixed delay, corresponding to the execution of control laws relating “pedal-press” event and “wheels-stop” event. We call this constraint a *latency* constraint. The word “latency” was already employed by the real-time community. For example, Hagmann [4] calls latency the necessary time to obtain data when reading a hard disk or Aggarwal [5] the necessary time of communicating data from memory to a processor. More theoretical definitions were given by Goddard [6] in the case of data-flow graphs or by Van Beek [7] in the case of repeated precedence constraints. These definitions are limited to particular cases and can not be employed to describe the elapsed time between any two operations related by precedence constraints. Obviously if a latency constraint

---

\*Corresponding author

should describe this time then this constraint is “end-to-end” constraint and the interest in these constraints was very well justified by Gerber [8]. Also, the examples given by Gerber are treating only “end-to-end” constraints between sensors and actuators.

Our paper generalizes the results concerning the end-to-end constraints [8] [9] [10], allowing to have several latency constraints which are not imposed only between sensors and actuators, but also between the operations which are necessary to obtain output events from input events. For example, it is possible to specify the brake system of a car (Figure 1) by a graph where *PRESS* represents a sensor giving the information which indicates that the driver presses the brake pedal, *SPEED* represents a sensor giving the wheel speed, *DEAC* represents the deactivation of the speed regulator, *ESTIM* represents the estimation of the pressure on the brake pedal, *ABS* represents the anti-locking brake system (ABS) and *BRAKE* represents the actuator giving the pressure on the brakes wheels. For example, we may need to impose a delay between *PRESS* and *BRAKE*, and another delay between *DEAC* and *BRAKE*.

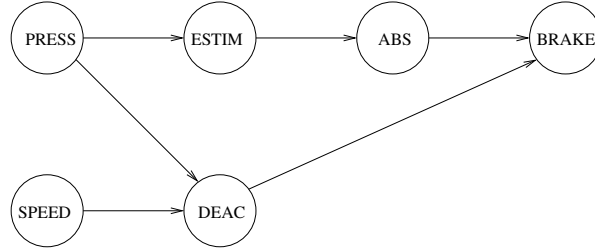


Figure 1: Example of brake control

The formal definition of our model is given in section 2. From now on our model is called latency-based model.

The results on non-preemptive deadline-based of real-time systems with release times, periodicity constraints and deadlines are poor [11]. Giving the results we have in the non-preemptive latency-based problem, this paper presents a connection between the two models allowing to use the latency-based problem in order to propose new results for the deadline-based problem.

This paper is composed of five sections. The following section introduces the two models. The section 3 gives an algorithm indicating between available operations which is the most important according to the deadlines of their successors. It is followed by the section 4 which proposes an algorithm transforming an instance of a scheduling problem using the deadline-based model into an instance of a scheduling problem using the latency-based model. The paper ends with section 5 which concludes and gives hints for future research.

## 2 Two real-time models

Before presenting the two models, we give the definition of a schedule and of the schedulability of a system of operations, where by system of operations we understand a set of operations with computation times and different constraints.

**Definition 1** *For a system of operations, a schedule  $S$  is a total order on the set of operations associating to each operation  $A$  a start time  $s_A$  which is the instant when the operation is executed. We denote by  $S$  the set of the possible schedules of a system of operations. An operation with the start time known is called scheduled.*

**Definition 2** *A system of operations is schedulable if there is at least one schedule satisfying the constraints of the system.*

We start by presenting the deadline-based model of operations given by Liu and Layland (see Figure 2) in the non-preemptive case. An operation  $A$  becomes available at its release time  $s_A$ . The release time is periodically repeated with a period  $T_A$ . So if the first release of an operation

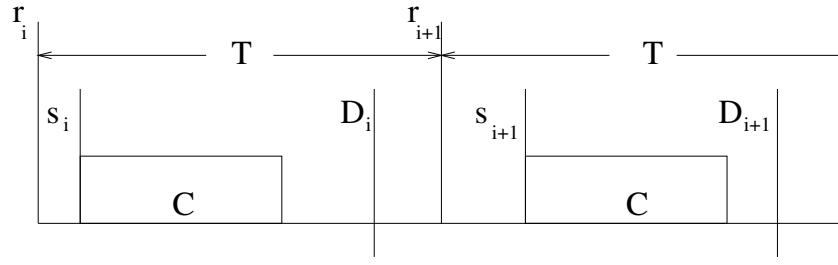


Figure 2: Model proposed by Liu and Layland

is  $r_0$  then the  $i^{th}$  release time of the operation will be  $r_A + (i - 1)T_A$ . Also the computation time  $C_A$  is known.

As said in the first section, in the latency-based model, the operations are always available and the start time  $s_A$  of an operation  $A$  is repeated periodically with a period  $T_A$  (see Figure 3). The computation time  $C_A$  is known.

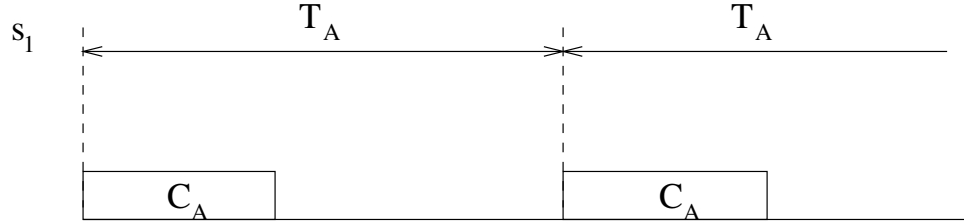


Figure 3: Model of strictly periodic operations without release time

In the deadline-based model the instant when an operation becomes available is always known (release time), but not the instant when it is scheduled. Knowing the start time of the first repetition of an operation does not imply knowing the start times of all the other repetitions of the operation.

In the latency-based model the instant when an operation becomes available and the instant when it is scheduled are unknown only for the first repetition of the operation. Once the first start time of an operation is known, the start times of all the other repetitions are also known and the release times of these repetitions are useless.

The difference between the two models is depicted by comparing them for some operation  $A$  in Figure 4. The first schedule is given for the deadline-based model and the second one is given for the latency-based model.

The latency-based model allows to impose two other constraints: the precedence constraints and the latency constraints. We use a graph-based model to specify the precedences between operations through an algorithm graph. The graph is a directed graph  $G = (V, E)$  where  $V$  is the set of operations and  $E \subseteq V \times V$  the set of edges which represents the precedence constraints possibly due to data transfer between operations. Therefore, the directed pair of operations  $(A, B) \in E$  means that  $B$  must be scheduled, only if  $A$  was already scheduled. When precedence constraints are defined, we say that an operation is available when all its predecessors are already scheduled [12].

The edges of the graph define a partial order on the set of operations. The operations which do not belong to an edge define a “potential parallelism” [13]. Moreover, because the graph describes a real-time system, which is reactive, this latter graph has a *pattern* infinitely repeated [14] which induces an infinite repetition of all operations (see Figure 5).

For the sake of simplicity and because we use only directed graphs, from now on we will use the term path instead of directed path.

We denote by  $\mathcal{P}$  the set of all paths of graph  $\mathcal{G}$  and we say that  $P(A, B) \in \mathcal{P}$ , if there is at least a path from  $A$  to  $B$ . If  $\nexists P(A, B) \in \mathcal{P}$ , then there is no path from  $A$  to  $B$ . Moreover, we denote

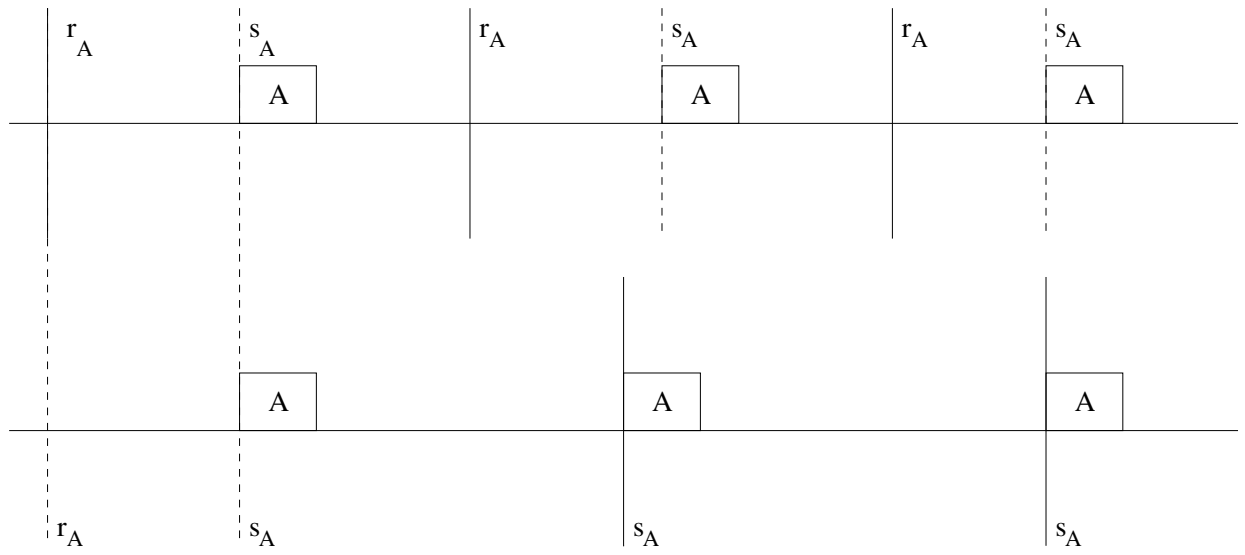


Figure 4: Difference between the deadline-based model (above) and the latency-based model

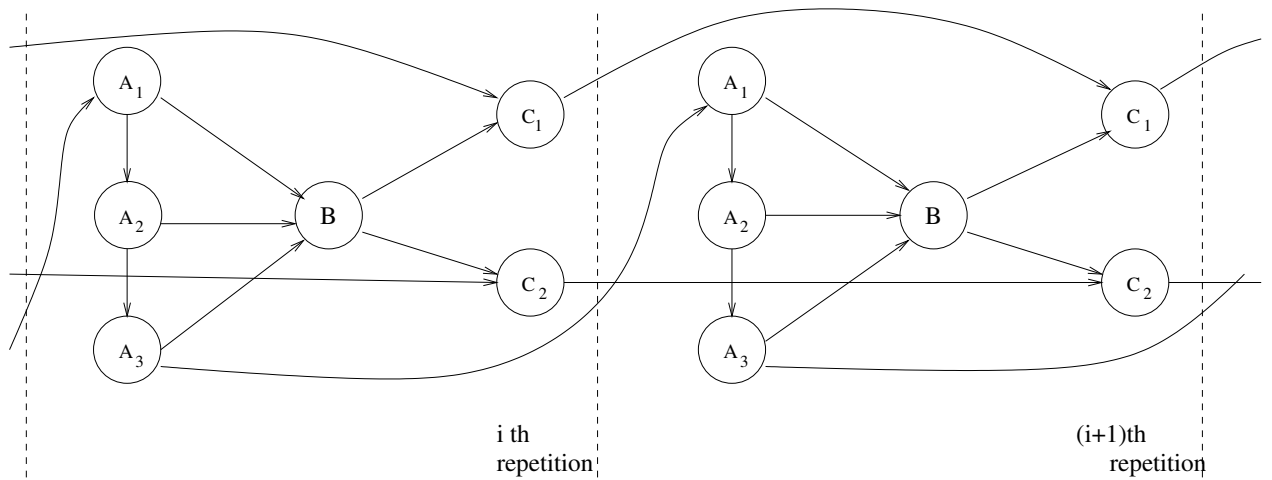


Figure 5: Graph describing a real-time system

by  $\mathcal{D}(A) = \{B \in V \text{ such that } P(A, B) \in \mathcal{P}\}$ .

**Definition 3** [3] *On a pair of operations  $(A, B)$  with  $\exists P(A, B) \in \mathcal{P}$  and  $A$  and  $B$  belonging to the same pattern, we say that a latency constraint  $L(A, B)$  is imposed if the operations are scheduled such that  $s_B + C_B - s_A \leq L_{AB}$ ,  $L_{AB} \in \mathbb{N}^+$ . Let  $\mathcal{L}$  be the set of all latency constraints imposed for a system.*

Without any loss of generality we assume that all characteristics, i.e. the computation times, the periods and the latency constraints, are defined as multiples of a clock tick  $\tau$  (time is discrete). Afterward the integer values of the given characteristics are implicitly multiplied by  $\tau$ .

### 3 Deadline-marking algorithm

In this section, we present a theorem proving that a latency constraint is a deadline. Thus we may solve the non-preemptive problem of systems with precedence, strict periodicity and deadlines. This problem is a case of latency-based problem with the latency constraints replaced by deadlines. The results of this section may be applied for both uniprocessor and multiprocessor cases.

The following theorem gives the relationship between the deadlines and the latency constraints.

**Theorem 1** *A latency constraint  $L(A, B)$  is a deadline for the operation  $B$ , once the operation  $A$  is scheduled.*

**Proof** We prove the theorem by double implication.

First, we demonstrate that a latency constraint  $L(A, B)$  of a pair  $(A, B) \in \mathcal{L}$  may be expressed as the deadline of operation  $B$ , once  $A$  is scheduled. Because  $(A, B) \in \mathcal{L}$ , then we have  $s_B \leq L_{AB} + s_A - C_B$ . If we denote by  $D_B$  the expression  $L_{AB} + s_A - C_B$ , we obtain

$$s_B \leq D_B \quad (1)$$

Once  $A$  is scheduled,  $s_A$  is known and  $D_B$  becomes known, also. Because  $s_A$  is calculated from the beginning of the schedule then the equation (1) defines a deadline for  $B$ .

Second, we express the deadline of an operation  $B$  as a latency constraint  $L(\cdot, B)$ . Because  $B$  has a deadline  $D_B$ , then  $s_B \leq D_B$ . We denote by  $A$  the first operation which was scheduled, then  $s_A = 0$ . Because the deadline is defined from the start of the schedule, we obtain that  $s_B - s_A \leq D_B$ . Moreover, because all the operations without predecessors may be the first scheduled operation, we have  $s_B - s_C \leq D_B, \forall C \in V$  with  $Prec(C) = \emptyset$  and  $B \in \mathcal{D}(C)$ , where  $Prec(C)$  is the set of predecessors of operation  $C$ . So, in order to satisfy the deadline of  $B$ , we define several latency constraints  $L_{CB} = D_B$  for each  $C$  such that  $Prec(C) = \emptyset$  and  $B \in \mathcal{D}(C)$ . The theorem is proved  $\square$

Due to this theorem, we may use the scheduling algorithm given in [3] for systems with precedence, strict periodicity and latency constraints to schedule systems with precedence constraints, strict periodicity constraints and deadlines. This algorithm uses a latency-marking algorithm. The Theorem 1 allows to modify this latter marking algorithm in order to take into account the deadlines. In this case the function *mark* of the operation  $A$  must be equal to the  $\min_{B \in \mathcal{D}(A)} \{D_B\}$ . These latter marks are obtained using the following algorithm, called deadline-marking algorithm:

#### Algorithm 1

Initialization:  $\mathcal{W} = \bigcup_{A \in V \text{ and } S_{uc}(A)=\emptyset} Prec(A)$  is the working-set. If  $A$  has a deadline  $D_A$ , then  $mark(A) = D_A$ , otherwise  $mark(A) = \infty$ .

Step 1: For  $A \in \mathcal{W}$ ,  $mark(A) = \min(mark(A), \min_{B \in S_{uc}(A)} \{mark(B)\})$  and  $\mathcal{W} = \mathcal{W} \setminus \{A\}$ . We add to  $\mathcal{W}$  the operations for which all the predecessors has been removed from  $\mathcal{W}$ .

Step 2: Repeat Step 1 until  $\mathcal{W} = \emptyset$ .

**Remark 1** *Recursively, each operation inherits the deadline of its successors. At the end of this latter algorithm, the mark of an operation may be equal to either its inherited deadline or its initial deadline. Therefore, the algorithm does not modify the marks of the operations without successors. This algorithm is, also, applied to the pattern.*

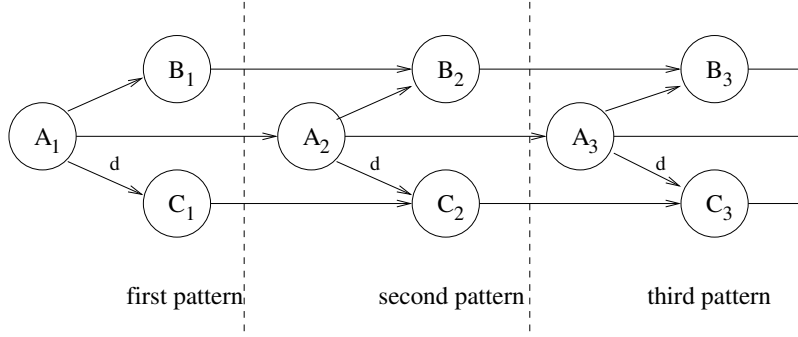


Figure 6: Graph of example 1

**Example 1** *Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a graph given in Figure 6 with the deadline  $D_{C_1}$ . In Figure 6, the letter  $d$  on the edge from  $A_1$  to  $C_1$  implies that the precedence constraint is due to data transfer. All the operations will have the mark  $D_{C_1}$  except  $C_2$ . The algorithm is given in table 1, where by  $m(A)$  we understand  $mark(A)$ .*

|                | $m(A_1)$  | $m(A_2)$  | $m(A_3)$  | $m(B)$    | $m(C_1)$  | $m(C_2)$  |
|----------------|-----------|-----------|-----------|-----------|-----------|-----------|
| Initialization | $\infty$  | $\infty$  | $\infty$  | $\infty$  | $D_{C_1}$ | $\infty$  |
| $D_{C_1}$      | $D_{C_1}$ | $D_{C_1}$ | $D_{C_1}$ | $D_{C_1}$ | $D_{C_1}$ | $D_{C_1}$ |

Table 1: Marks obtained using the marking algorithm

## 4 Transformation from deadline-based problem to latency-based problem

In this section, we give an algorithm transforming an instance of a scheduling problem using the deadline-based model into an instance of a scheduling problem using the latency-based model. We prove that the two problems are equivalent from the complexity point of view. Thus, the deadline-based problem may be solved in the non-preemptive case using the results for the latency-based problem. The results of this section may be applied for both uniprocessor and multiprocessor cases.

We consider a system with  $n$  operations and for each operation  $A$ , we have the computation time  $c_A$ , the release time  $r_A$  and the deadline  $D_A$ . The operation  $A$  is periodic with the period  $T_A$ . We transform this system using the following algorithm:

### Algorithm 2

Step 1: for each operation  $A$  we add a fictive operation  $A'$  with the computation time equal to 0.

Step 2: we add a precedence constraint between  $A'$  and  $A$ . We obtain the graph of precedence constraints with  $2n$  operations and  $n$  edges.

Step 3: we impose latency constraints for each  $(A', A)$  with  $L_{A'A} = D_A$ .

Step 4: we impose strict periodicity constraints for each operation  $A'$  with  $T'_A$  equal to the period of operation  $A$ .

**Example 2** This example illustrates the Algorithm 2. For a system of three operations  $A$ ,  $B$  and  $C$  with no precedence constraints, we obtain the graph given in Figure 7.

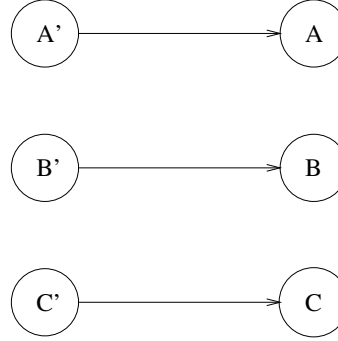


Figure 7: Graph obtained using the Algorithm 2

We denote by  $PRE$  the decision problem associated to the scheduling problem of periodic systems with deadlines. We denote by  $PPLc$  the decision problem associated to the scheduling problem built from  $PRE$  using the Algorithm 2.

**Theorem 2** The decision problem  $PRE$  has, as solution, the answer “yes” if and only if the decision problem  $PPLc$  has, as solution, the answer “yes”.

**Proof** We prove the theorem by double implication.

For the decision problem  $PRE$  each operation  $A$  has a release time  $r_A$  which gives the instant when the operation becomes available, a deadline  $D_A$  defined from the release time, a period  $T_A$  and a computation time  $C_A$ .

First, we suppose that the problem  $PRE$  has the answer “yes” and we prove the decision problem  $PPLc$  has also the answer “yes”. We denote by  $S$  one of the schedules which satisfies the constraints of the scheduling problem  $PRE$ . This implies that for each operation  $A$  we know the start time  $s_A \in S$  such that  $r_A \leq s_A \leq D_A$ .

From the schedule  $S$ , we build a schedule  $S'$  for the problem  $PPLc$ . Each operation  $A'$  associated to an operation  $A$  of an instance of the problem  $PRE$  has the start time of its first repetition  $s'_{A'_1}$  equal to  $r_A$ . Each operation  $A$  of an instance of the problem  $PPLc$  has the start time equal to the start time of the operation  $A$  in the schedule  $S$ . The Figure 8 illustrates this construction. The operation  $A'$  with the computation time equal to 0 is marked with non-continuous line on the figure. Consequently, because the deadlines are satisfied in the schedule  $S$  then all the latency constraints are satisfied in the schedule  $S'$ . Because the operations  $A'$  has the computation times equal to 0 and their start time periodically repeated due to the periodicity of release times in the schedule  $S$ , then the periodicity constraints are satisfied in the schedule  $S$ .

Second, we build similarly a schedule for the problem  $PRE$  from a schedule of the problem  $PPLc$ . Consequently, there is at least a schedule for the first problem if and only if there is a schedule for the problem built using the Algorithm 2. The theorem is proved  $\square$

We expressed the scheduling problem with release times, periodicity constraints and deadlines using the latency-based model.

## 5 Conclusion and further research

We compare two real-time systems models in the non-preemptive case. We first give an algorithm which allows us to solve a particular case of the latency-based problem, the scheduling problem

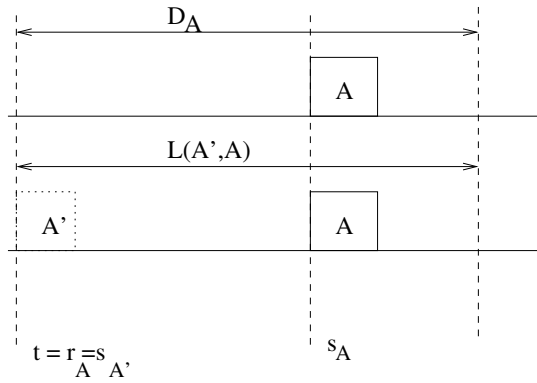


Figure 8: Construction of a schedule for  $PPLc$  from a schedule for  $PRE$

with precedence, strict periodicity constraints and deadlines. Moreover we give an algorithm which transforms an instance of the deadline-based problem into an instance of the latency-based problem. This allows to particularize the latency-based scheduling algorithm to the deadline-based problem.

We want to extend these results to the non-preemptive in the multiprocessor case, where the results for the deadline-based problem are also poor.

## References

- [1] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [2] J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra. Scheduling periodic tasks. *INFORMS Journal on Computing* 8, 1996.
- [3] L. Cucu, R. Kocik, and Y. Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. In *Proceedings of Real-time and Embedded Systems*, 2002.
- [4] R.B. Hagmann. Low latency logging. Technical report, Palo Alto Research Center, 1991.
- [5] A. Aggarwal and A.K. Chandra. On communication latency in pram computations. Technical report, IBM Research Division, 1989.
- [6] S. Goddard. *Constraints on data-flow*. PhD thesis, University of North Carolina, 2000.
- [7] Peter van Beek and Kent D. Wilken. Fast optimal instruction scheduling for single-issue processors with arbitrary latencies. In *CP*, pages 625–639, 2001.
- [8] Richard Gerber, Seongsoo Hong, and Manas Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Trans. Software Eng.*, 21(7):579–592, 1995.
- [9] D.-I. Kang, Gerber R., and M. Saksena. Performance-based design of distributed real-time systems. *Proceedings of IEEE Real-Time Technology and Applications*, 1997.
- [10] Richard Gerber, William Pugh, and Manas Saksena. Parametric dispatching of hard real-time tasks. *IEEE Trans. Computers*, 44(3):471–479, 1995.
- [11] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. *IEEE*, pages 129–139, 1991.
- [12] L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. In *Proceedings of Real-time and Embedded Systems*, 2003.



- [13] Y Sorel. Massively parallel computing systems with real time constraints, the "algorithm architecture adequation" methodology. In *Proceedings of Massively Parallel Computing Systems, Italy*, 1994.
- [14] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real time embedded heterogeneous multiprocessors. *Codes'99 7th International Workshop on Hardware/Software Co-Design*, 1999.

# A Comparison of Combinatorial Optimization and Dispatch Rules for Online Scheduling

David Montana  
BBN Technologies  
10 Moulton Street, Cambridge, MA 02138  
EMail: dmontana@bbn.com

## Abstract

Online, or dynamic, scheduling refers to when the tasks to perform are not known prior to the execution of the schedule but rather are continually revealed during the execution process. The real-time aspect of online scheduling potentially changes the preferred properties of the scheduling algorithm. In particular, the time required to generate a schedule becomes a more important factor, while the quality of the schedules produced becomes less important. The relative importance of these two properties, schedule creation time and schedule quality, differs greatly depending on the characteristics of the scheduling problem. Here, we describe an empirical investigation of how the tradeoff between them varies with the scheduling problem. We do this by comparing the performance of an optimizing scheduler, which creates better schedules, and a dispatch-rule scheduler, which provides faster turnaround, on a set of different online scheduling problems. Our experiments identify some key factors in determining which type of scheduler is better: (i) the predictability with which schedules are executed and (ii) the time scale with which the scheduling problem changes as compared to the time required to optimize a schedule.

## 1 Introduction

For online scheduling, also referred to as dynamic scheduling, schedule creation and schedule execution are not separate processes, but rather are inextricably intertwined. The tasks are not known before starting the execution process but rather incrementally arrive in the scheduler's task "bin" during execution [6]. Hence, schedules need to be dynamically created and updated in real time.

Online scheduling places different requirements on a scheduling algorithm. One such requirement is that schedules should be created and updated quickly enough to react to changes dynamically. The consequence is that

often a different type of algorithm is required for online scheduling than "standard" static scheduling. For example, to schedule processes in a computer operating system, it does not make sense to use combinatorial optimization, because any decision needs to be made too quickly. Instead, a simple dispatch rule for when to switch processes and which process to run next is better.

However, not all online scheduling problems are like this. For example, field service scheduling (i.e., scheduling repairpeople to service calls) is an online scheduling problem, but an optimizing scheduler will work well for this problem [9]. The reasons for the difference, as we will discuss, are a larger time scale and schedule execution that is relatively predictable.

In this paper, we carry out a systematic experimental exploration of the tradeoff in importance between schedule creation time and schedule quality as the properties of the online scheduling problem vary. To do this, we define a sample online scheduling problem that is simple enough that it can be easily understood and analyzed but complex enough that scheduling performance can potentially benefit from combinatorial optimization. We implement the dynamic aspect of the problem as a simulation in pseudo-real time. This allows explicit control over the time for schedule creation relative to the actual time of the process scheduled.

Parameters control various aspects of the online scheduling problem, including a variety of statistical distributions, such as the arrival time of tasks, the time to complete tasks, and the requirements on when to complete tasks. Also determined by parameters are precedence relationships, delays, transition times, and priorities. Varying these parameters changes the nature of the scheduling problem.

We use two different schedulers. One performs combinatorial optimization; since the problem is sufficiently small, it finds the optimal schedule according to our defined criterion. A second performs dispatch-rule scheduling, optimizing the same criterion but only in a greedy manner as resources become free. This allows

comparison between the two types of scheduling as we vary the nature of the scheduling problem as well as the schedule creation times.

The rest of the paper is structured as follows. Section 2 discusses previous work on online scheduling and related topics. Section 3 defines our sample scheduling problem in detail. Section 4 describes the two different scheduling algorithms used. Section 5 discusses experiments and results.

## 2 Previous Work

While it is still prevalent to treat scheduling as a static optimization problem, there has been a variety of work on the dynamic aspects of scheduling. We now examine different threads of research on dynamic scheduling. [Note that we only cite representative examples; more comprehensive surveys of the field are available, such as [15]. Further note that the vocabulary in the field is not standardized, with researchers often using terms differently. We attempt to be as consistent as possible with previous usage.]

One general approach, used by [4] and [14] among others, divides the scheduling process into planning and execution phases. During the planning phase, a *predictive scheduler* performs combinatorial optimization to create a nominal schedule. During the execution phase, a *reactive scheduler* continually modifies or repairs the nominal schedule to account for unanticipated circumstances. One goal of the reactive scheduler is to keep the schedule as close as possible to the original nominal schedule. This approach makes sense when all the scheduling data (including what tasks to schedule and the state of the resources) is known for a certain period of time into the future, and deviations that arise during execution are likely to be minor.

A second approach to dynamic scheduling is *dispatch rules*. In practice, the use of dispatch rules is very common, although the choice to use dispatch rules for real-world scheduling often has more to do with ease of implementation than scheduling performance. There is a large variety of different rules that have been used, many of them chronicled in summaries such as [11]. When simple heuristic rules, such as earliest-due-date (EDD) and first-in-first-out (FIFO), were applied to a range of different scheduling problems, it was found that which rule was best varied with the problem [10]. There is also the potential for using a set of rules, rather than a single rule, in an expert system [7]. The advantage of using dispatch rules is that it enables quick reaction to changes. The disadvantage is that it creates suboptimal schedules because it does not plan into future. This approach for online scheduling is best when there is online

scheduling (i.e., tasks being revealed during execution), the fundamental time scale of the problem (a concept we will examine below) is short, and the future is very uncertain.

A third approach to dynamic scheduling is continuous combinatorial optimization. Periodically, a scheduling algorithm essentially recomputes the entire schedule to reflect the changed situation. A simple repair algorithm changes the schedule minimally as required to ensure that no hard constraints are violated due to unexpected deviations from the schedule between scheduler updates. In practice, the results can be similar to the predictive/reactive approach. This is particularly true when the optimization criterion includes a term that rewards schedule stability, i.e. keeping the schedule as similar to the previous schedules as possible, usually for the sake of humans either being scheduled or assisting with the schedules. (Examples of the use of a schedule stability terms are given in [12] and [13].) The fundamental distinction between this approach and the predictive-reactive approach is that the latter has two distinct scheduling algorithms (one for planning and one for execution) while the former uses a single scheduling algorithm (with no distinction between planning and execution). A real-world example of this approach applied to field service scheduling, i.e. scheduling of repair people, is described in [9]. This is a good approach when there is online scheduling, the future is relatively predictable (at least in the near term), and the time scale is long enough that an optimized schedule can be generated within a time that the problem is not likely to change significantly.

The fourth approach is stochastic optimization. Algorithms of this variety perform a statistical analysis of possible future scenarios (usually via a Monte Carlo method) to compare expected payoffs of different scheduling decisions. Examples of this approach are [5], [2] and [3]. This approach works well when there is online scheduling and the future is unpredictable in a deterministic sense but is predictable in a statistical sense. Except in those cases where strategies can be precomputed, this approach (like combinatorial optimization) requires that the fundamental time scale of the scheduling problem not be too small.

As we have pointed out, these different approaches would be expected to perform well under different conditions. Certain extreme situations are clearly suited to one particular approach. (For example, the very short time scale of operating system scheduling means that dispatch rules are best suited.) However, often the situation is in some middle ground, and it is not clear which approach is best. In this paper we examine how to quantify the tradeoffs to help decide on the best approach.

### 3 Test Problem Definition

In order to evaluate the performance of different online scheduling algorithms, we need one or more test problems. We have invented a test problem with the following desirable properties:

- It involves online scheduling.
- There is a clear potential advantage to planning into future, but also a clear potential advantage to fast schedule turnaround (hence establishing the fundamental tradeoff).
- Altering the parameters specifying the statistical properties of the data changes the nature of the problem (essentially allowing the creation of different scheduling problems) in a well controlled way.

We now describe the details of this test problem.

#### 3.1 Jobs, Tasks and Resources

A *task* is an atomic unit of work to perform. A *job* is a set of one or more interrelated tasks. Each task is part of exactly one job. The tasks within a job form a simple sequence, with the first task in the sequence needing to finish before the second begins, and so on. There is potentially a non-zero delay between tasks in a sequence, where a task must wait a specified amount of time after its preceding task completes before being allowed to execute.

Jobs enter the system continually. Each job has associated with it three times:

- arrival time: when the system/scheduler first has knowledge of the job
- release time: the earliest that the job can start executing
- deadline: the time by which the job must complete or else be considered late

Each job is a member of a *job class*. All the jobs in a particular job class share the following properties:

- (fixed) sequence of tasks with the associated delays
- (fixed) utility, which is the cost of not finishing the job within the deadline (with the overall performance metric being the sum over all dropped jobs of the job's utility)

All the jobs within a job class are generated probabilistically from the following statistical distributions:

- jobs arrive following an exponential distribution, specified by the mean time between arrivals
- each job's holding duration (time between arrival and release) is generated from a truncated normal distribution, specified by the mean, standard deviation, and minimum value
- each job's execution duration (time between release and deadline) is generated from a truncated normal

distribution

A *resource* can perform at most one task at a time. Each resource can perform only tasks of a certain *task type*, where a task's type is defined by its job's class and its position in the sequence of the job. The time required to execute a task, which depends on the task type, is generally non-deterministic. Between tasks of different types, there are potentially non-zero resource transition times, during which the resource is not available to execute a task. Resources fail intermittently and are not available to execute tasks until they are repaired.

Each resource is a member of a *resource class*. All the resources in a resource class have the same properties, which include:

- which task types are handled
- (fixed) transition times between tasks of different types as a function of the two task types
- truncated normal distributions of task execution times, with a different distribution per task type, each determined by a mean and standard deviation with a minimum value of 1
- exponential distributions of time between failures and time for repairs, determined by the mean values

Unlike jobs, all resources are present at the start of the simulation and persist throughout. Each resource class specifies:

- the number of resources of that class

#### 3.2 Pseudo-Real-Time Execution

In the previous subsection, we examined the static aspects of the scheduling problem, and now we discuss the dynamic aspects. To perform online scheduling, we need a dynamic process with a sense of time progressing. In this case, we create a simulated dynamic process with a simulation clock, which provides us with easy control over time-related issues.

There are three basic processes occurring in the simulation:

1. Events (associated with new job arrivals, resource failures, and resource repairs) are arriving into the system.
2. The scheduling algorithm is creating schedules by assigning tasks to resources at specified future times.
3. Tasks are being executed by resources as specified by the current schedule.

**Event Processing** - The simulation maintains a list of jobs with their arrival times invisible to the scheduling and execution processes. When the simulation time advances to the arrival time of a job, the simulation injects the job into the system by revealing it to the scheduling and execution processes. Resource failure events and re-

source repair events are treated similarly. Note that the simulation time is quantized so that every event must occur at a time that is an integral number of simulated seconds into the run.

**Scheduling** - The scheduling algorithm is invoked to create a new schedule under three possible conditions:

- a new event (job arrival, resource failure/repair) occurs
- a specified number of seconds has passed since the last invocation of the scheduler
- a reactive (dispatch-rule) scheduling algorithm is being used **and** a resource becomes free

However, if the scheduler is already in the simulated process of creating a schedule when cued to create a new schedule, the new schedule creation process must wait to begin until the previous schedule is completed.

The scheduling algorithm requires a potentially non-zero amount of simulation time to make its decisions and create a new schedule. The number of seconds required for the scheduler to run is a parameter of the simulation. (Note that the simulation time required to generate a schedule has no relation to the actual time that the algorithm runs.) During the simulated time that the scheduler is creating a schedule, it ignores any changes to the underlying data that occur. These changes include not only new events but also deviations from expected task execution times. Furthermore, the schedule that was last created stays in effect for the duration of the simulation time required to create a new schedule. Note that this schedule creation duration can cause degraded utilization of the resources because: (i) there is a delay in reacting to changes and (ii) newly created schedules are based on outdated information.

**Task Execution** - One important aspect of task execution is the decision which task to execute with which resource at what time. If everything goes according to the schedule, then each resource executes each task assigned to it at the time when it is assigned. However, due to the stochastic nature of the problem, generally schedules do not proceed exactly as planned, and some form of schedule repair is required. The execution process does the simplest repair to ensure that the schedule does not violate hard constraints (leaving any more sophisticated adjustments to the schedule to the scheduling process). Each resource maintains the same set of assigned tasks in the same order that they are scheduled, but ignores the absolute times of these assignments. The resource executes the next task at the earliest time that does not violate hard constraints (including task precedence relationships, earliest release time, and resource failures and transition periods). Note that this means a task can be executed earlier than it was scheduled (if the preceding task in its sequence or the preceding task on its resource finishes earlier than anticipated) as well as later than it

was scheduled.

A second aspect of task execution to consider is how to handle resource transition periods, i.e. the time required for a resource to reconfigure to handle a task of a different type than the previous one. We specify that a resource should start the transition process at the earliest possible time that it appears a transition will be needed and the resource is idle. It should not wait until the next task is ready to execute, because that will cause an unnecessary delay in the tasks's execution waiting for the transition to occur. There will be times, namely when the schedule changes which task to execute next, when this policy can potentially cause extra delay, but overall it leads to less time with resources idle.

A third aspect of task execution is what happens to tasks and transition periods that are in the middle of executing when a resource fails. We specify that any such interruption of a task or transition has the effect that not only does the task or transition stops executing but also that the state is as if the task or transition had never started executing. Furthermore, such a task is taken off the schedule until the scheduling process puts it back on the schedule.

## 4 Scheduling Algorithms

The way we have defined our scheduling problem, there will generally be clear distinctions between well-performing and poorly-performing schedulers. The good schedulers will align tasks so as to minimize the transition and idle times of the resources. They will also make good decisions, when overwhelmed with tasks, about which jobs to drop and will make those decisions early in the jobs' task sequence to avoid wasted execution time. This will result in more jobs of higher utility finishing by their deadline.

We have implemented two scheduling algorithms, whose performance we will compare empirically. One algorithm uses combinatorial optimization, while the other employs a dispatch rule. We do not claim that these particular algorithms are the best of their respective classes in terms of performance, but they are representative. Hence, we can draw at least preliminary conclusions about the relative merits of the two classes of scheduling algorithms.

We now describe each of these two algorithms.

### 4.1 Optimizing Scheduler

We have created the optimizing scheduler using Vishnu [8, 1]. Vishnu is a scheduling application we developed that is what we call a *reconfigurable scheduler*, which means that it can be configured for a wide variety of dif-

ferent scheduling problems without modifying the software. For the purposes of this work, we configured it to implement the hard constraints described above and the optimization criterion given below. It will search for, and generally find (given enough time), the optimal schedule that satisfies the hard constraints. (The actual time required to perform the optimization does not affect the simulation results, since simulation time is completely separate from wall-clock time.)

The optimization criterion for the scheduler should consider not just how a schedule as constituted satisfies the criterion of completing jobs on time, but also what happens beyond its time horizon and how future events could disrupt the schedule. We define the optimization criterion as

$$\sum_{j \in J} \rho(j) \cdot P_f(j)$$

where  $J$  is the set of all outstanding jobs (i.e., jobs that are neither completed nor overdue),  $\rho(j)$  is the utility of job  $j$ , and  $P_f(j)$  is the probability that  $j$  does not complete on time given the proposed schedule.

The value of  $P_f(j)$  must be estimated, even for jobs whose deadlines are within the scheduling window, because of the stochastic nature of the task execution process. (Note that the scheduling window is the finite interval of time into the future into which the scheduler can place tasks. For a job whose deadline lies within this window, we know whether or not the job is scheduled to finish in time, but can only estimate whether it will actually finish in time.) We do not use a formal approach since there is no analytic solution and a Monte Carlo approach would take too long given all the jobs in all the schedules we need to handle. Instead, we just use a reasonable, quick estimate (in a similar fashion to how chess-playing algorithms use a quick board evaluator to evaluate one of many potential paths into the future).

The estimate of  $P_f(j)$  for job  $j$  is computed as follows. Let  $t_s(j)$  denote the last task in the sequence of tasks for  $j$  that is scheduled if such a task exists. (It will not exist if no tasks of  $j$  are currently scheduled.) Let  $t_n(j)$  be the first task in the sequence of tasks for  $j$  that is not scheduled if such a task exists. (It will not exist if all tasks of  $j$  are currently scheduled.) Either  $t_s(j)$  or  $t_n(j)$  must exist, and if they both exist then they are consecutive. If  $t_s(j)$  exists, compute the quantity

$$\text{Prob}\{E_{req}(j) < \text{rand}(E_{sch}(t_s(j)) + T_{foll}(t_s(j)), \sigma_{foll}(t_s(j)) + \sigma_{exec}(t_s(j)))\} \quad (1)$$

where  $E_{req}(j)$  is the required end time (i.e., deadline) for  $j$ ,  $E_{sch}(t_s(j))$  is the scheduled end time for  $t_s(j)$ ,  $T_{foll}(t_s(j))$  is the expected time to complete job  $j$  after finishing task  $t_s(j)$ ,  $\sigma_{foll}(j)$  is the standard deviation in

the time to complete  $j$  after finishing  $t_s(j)$ ,  $\sigma_{exec}(j)$  is the standard deviation in the task execution time of  $t_s(j)$ , and  $\text{rand}(m, \sigma)$  is a random number selected from a normal distribution with mean  $m$  and deviation  $\sigma$ . If  $t_n(j)$  exists, compute the quantity

$$\text{Prob}\{E_{req}(j) < \text{rand}(E_{win} + T_{incl}(t_n(j)), \sigma_{incl}(t_n(j)))\} \quad (2)$$

where  $E_{win}$  is the end time of the scheduling window,  $T_{incl}(t_n(j))$  is the expected time to complete  $t_n(j)$  and any subsequent tasks in  $j$  (with the caveat that once the window is shifted forward in time  $t_n(j)$  could actually be scheduled to start earlier than the previous window end time without anything else changing), and  $\sigma_{incl}(t_n(j))$  is the standard deviation in the time to complete  $j$  including  $t_n(j)$ . Then,  $P_f(j)$  is given by Equation 1 if  $t_n(j)$  does not exist, is given by Equation 2 if  $t_s(j)$  does not exist, and is the average of these two terms otherwise.

One detail we passed over is how we estimate quantities such as  $T_{foll}(t)$ . We posit

$$T_{foll}(t) = \sum_{t_f \in T_f} (T_{delay}(t_f) + T_{exec}(t_f) + T_{idle}(t_f))$$

where  $T_f$  is the set of all tasks following  $t$  in sequence,  $T_{delay}(t_f)$  is task precedence delay time,  $T_{exec}(t_f)$  is estimated task execution time and  $T_{idle}(t_f)$  is the time beyond the minimum that  $t_f$  has to wait to obtain a resource. While we know  $T_{delay}$  and  $T_{exec}$  a priori, we just make a guess (with constant set via experimentation) that  $T_{idle}$  is approximately equal to  $T_{exec}$ . So,

$$T_{foll}(t) = \sum_{t_f \in T_f} (T_{delay}(t_f) + 2.0 \cdot T_{exec}(t_f))$$

Similarly, we estimate

$$\sigma_{foll}(t) = \sum_{t_f \in T_f} (1.4 \cdot T_{exec}(t_f))$$

$$T_{incl}(t) = T_{foll}(t) + 1.7 \cdot T_{exec}(t)$$

$$\sigma_{incl}(t) = 0.7 \cdot T_{incl}(t)$$

## 4.2 Dispatch-Rule Scheduler

The dispatch-rule (reactive) scheduler does no look-ahead in terms of planning schedules, instead waiting for a resource to become free before choosing a single task to assign this resource. The general approach for a dispatch rule is to define a score associated with assigning a given task to a given resource and select the eligible task that minimizes (or maximizes) that score for

the chosen resource. For our dispatch rule, we have defined a scoring function that uses the same basic idea as the optimization criterion for our optimizing scheduler. Not only does this work effectively, but it also reduces the differences between the two schedulers beyond the fundamental difference, the amount of look ahead.

For resource  $r$  and task  $t$  in job  $j$ , we define the score for  $t$  as

$$\rho(j) \cdot (P_{assign}(t) - P_{not}(t))$$

where  $P_{assign}(t)$  is the probability that  $j$  fails to complete on time if  $t$  is assigned to  $r$  and  $P_{not}(t)$  is the probability that  $j$  fails to complete on time if  $t$  is not assigned to  $r$ . This will usually be a negative number, with a more negative value indicating a bigger difference made by scheduling  $t$  now rather than waiting. Essentially, this is doing triage, throwing away those jobs that are unlikely to finish on time and postponing those jobs that can safely be postponed. We estimate  $P_{assign}(t)$  just as in in Equation 1 with  $t_s(j) = t$

$$\text{Prob}\{E_{req}(j) < \text{rand}(E_{sch}(t) + T_{foll}(t), \sigma_{foll}(t) + \sigma_{exec}(t))\}$$

We estimate  $P_{not}(t)$  as

$$\text{Prob}\{E_{req}(j) < \text{rand}(E_{sch}(t) + T_{foll}(t) + T_{exec}(t) - T_{idle}(t), \sigma_{foll}(t) + \sigma_{exec}(t))\} \quad (3)$$

where in this case  $T_{idle}$  is the time between the current time and the start of any transition for  $t$ . Underlying Equation 3 is the assumption that  $t$  would be scheduled immediately after the conclusion of a similar type of task that started immediately.

## 5 Experiments

### 5.1 Datasets

As discussed in Section 3, there are a variety of parameters that specify the nature of the data, much of it in a statistical fashion. A listing of these different parameters is given in Table 1. We have developed a data generator that can create datasets with the properties specified by the parameters. Generally, because of the probabilistic nature of the data, each time the data generator is run with a given set of parameters it will result in datasets that are markedly different, albeit with the same statistical properties. To gain repeatability in running the simulation, we have developed the capability for the data generator to save and then replay dataset instances. For experimentation, this repeatability is important because it greatly reduces the “noise” in the experiments. With the different scheduling algorithms all working on the exact

same data, differences in performance reflect more accurately differences in the quality of the schedules generated.

We have created two different datasets, which we refer to as *predictable* and *unpredictable*. As shown in Table 1, they are the same in many ways, but there are some critical differences (highlighted in bold). One similarity is that they both have two job classes, JC1 and JC2, with jobs from JC1 consisting of a sequence of three tasks and those from JC2 consisting of a single task. Both datasets also have two resource classes, RC1 and RC2, with resources from RC1 handling the first task of jobs from JC1 as well as the tasks from JC2 and resources from RC2 handling the last two tasks of jobs from JC1. In both datasets, the resources from RC2 require two seconds to transition between the two different types of tasks they handle.

The key differences that make the unpredictable dataset less predictable are

- higher resource failure rate: Resource failures represent a large unanticipated glitch in the schedule. The tasks assigned to the failed resource can no longer run as scheduled, affecting not only these tasks but other tasks that depend on these via a “ripple effect”.
- greater variance of task execution times: The completion times of tasks will often be different from the expected end time. Again, this has ripple effects for other tasks on the same resource and other tasks in the same job.
- shorter holding and execution durations: With less advance warning about upcoming tasks and a greater urgency to start tasks as soon as possible, there is less time to plan in advance. This makes it more likely that new tasks will need to fit into the middle, rather than at the end of, the existing schedule and disrupt it.

The other differences between the two datasets are an attempt to keep the inherent level of difficulty of the two datasets approximately the same. Most notably, the increase from 2 instances of resource class RC2 (which is the bottleneck) to 3 instances going from the predictable dataset to the unpredictable dataset is a way to compensate for the fact that the resources are available less and are harder to allocate efficiently.

Both datasets are ten simulated hours long, providing roughly 5000-6000 instances of each job class.

### 5.2 Results

To evaluate the performance of a given scheduling algorithm on a given dataset, the key statistic we measure is the fraction of jobs finished on time. We divide this into separate measures for the two different classes of jobs, JC1 and JC2, since each of these job classes provides different challenges. JC1, with its three-task sequence

Table 1: Parameter Values for the Predictable and Unpredictable Datasets

| <i>Parameter</i>          | <i>Value 1 (Predictable)</i>               | <i>Value 2 (Unpredictable)</i>   |
|---------------------------|--------------------------------------------|----------------------------------|
| Resource Classes          | RC1, RC2                                   | RC1, RC2                         |
| Number of Resources       | <b>RC1: 3; RC2: 2</b>                      | <b>RC1: 3; RC2: 3</b>            |
| Task Types Handled        | RC1: TT1; RC2: TT2, TT3                    | RC1: TT1; RC2: TT2, TT3          |
| Mean Task Execution Time  | RC1: 2; RC2: 4, 6                          | RC1: 2; RC2: 4, 6                |
| Std. Dev. Task Execution  | <b>RC1: 0; RC2: 0, 0</b>                   | <b>RC1: 1; RC2: 2, 3</b>         |
| Resource Transition Times | TT2 → TT3, TT3 → TT2: 2                    | TT2 → TT3, TT3 → TT2: 2          |
| Mean Time Between Fails   | <b>RC1, RC2: <math>5 \cdot 10^8</math></b> | <b>RC1, RC2: 150</b>             |
| Mean Time For Repair      | RC1, RC2: 5                                | RC1, RC2: 5                      |
| Job Classes               | JC1, JC2                                   | JC1, JC2                         |
| Mean Time Bet. Arrivals   | <b>JC1: 5.7; JC2: 8</b>                    | <b>JC1: 6.5; JC2: 4</b>          |
| Min Holding Time          | <b>JC1: 15; JC2: 4</b>                     | <b>JC1: 0; JC2: 0</b>            |
| Mean Holding Time         | <b>JC1: 3; JC2: 4</b>                      | <b>JC1: 0.5; JC2: 0.5</b>        |
| Min Time to Complete      | <b>JC1: 35; JC2: 2</b>                     | <b>JC1: 25; JC2: 5</b>           |
| Mean Time to Complete     | <b>JC1: 45; JC2: 4</b>                     | <b>JC1: 30; JC2: 6</b>           |
| Dev. Time to Complete     | <b>JC1: 5; JC2: 1</b>                      | <b>JC1: 2; JC2: 1</b>            |
| Priorities                | JC1: 1; JC2: 2                             | JC1: 1; JC2: 2                   |
| Task Type Sequence        | JC1: TT1 → TT2 → TT3<br>JC2: TT1           | JC1: TT1 → TT2 → TT3<br>JC2: TT1 |
| Precedence Delay Times    | JC1: 0, 2, 1; JC2: 0                       | JC1: 0, 2, 1; JC2: 0             |

and longer holding and execution times, places more emphasis on planning. JC2, with just one task and quick required turnaround, puts a bigger emphasis on reaction time.

In addition to using different datasets and different scheduling algorithms, a third aspect we can vary is the schedule creation time. For the experiments, we only used the dispatch-rule scheduler with schedule creation time equal to zero. The reasoning here is that the benefit of the dispatch-rule scheduler is its ability to make fast decisions, so we assume that the time required for the decision process is insignificant. In contrast, we used a variety of different schedule creation times for the optimizing scheduler. This allowed us to explore a range of possibilities for schedule creation time to understand how it affects scheduling performance.

For all the runs with the optimizing scheduler, the scheduling window was set to 30 seconds.

The results are shown in Table 2. The delay is the schedule creation time.

A few observations about this data follow:

- When the scheduler delay (i.e., schedule creation time) is zero, the optimizing scheduler outperforms the dispatch-rule scheduler. This confirms our intuition that, all other things being equal, the ability of the optimizing scheduler to plan ahead gives it an advantage over a reactive scheduler.
- For zero delay, the difference between the optimizing and dispatch-rule schedulers is much greater for the predictable data than the unpredictable data. This

confirms our intuition that planning ahead loses much of its benefit when the future deviates significantly from what has been planned.

- As the simulated delay for the optimizing scheduler increases, the performance of this scheduler monotonically decreases. The dropoff in performance is much faster for the unpredictable data ( 15% dropoff for a 3 second delay) than for the predictable data ( 15% dropoff for a 10 second delay). This confirms our intuition that scheduling delay is more harmful with less predictable data.
- The decline in performance is also much steeper for jobs of class JC2 than for jobs of class JC1. This confirms our intuition that scheduling delay is more harmful for jobs that require quick turnaround.

## 6 Conclusion

There are two basic causes of schedule disruption in on-line scheduling. One is the schedule not executing according to plan, e.g. due to unexpected task execution times or resource failures. The other is the arrival of new tasks with constraints dictating that they must be placed in the middle, rather than at the end of, the schedule. In response to schedule disruption, it is best to react quickly because a delay in reaction results in a decrease in scheduling performance. This leads to an inherent tradeoff between planning carefully and reacting quickly, since careful planning takes time that delays the response.



Table 2: Percentage of jobs completed on time for the different scenarios

| <i>Scheduler</i>           | <i>Predictable</i> |            | <i>Unpredictable</i> |            |
|----------------------------|--------------------|------------|----------------------|------------|
|                            | <i>JC1</i>         | <i>JC2</i> | <i>JC1</i>           | <i>JC2</i> |
| Dispatch-Rule (no delay)   | 85.8               | 99.3       | 89.7                 | 99.6       |
| Optimizing (no delay)      | 94.2               | 100.0      | 92.9                 | 99.8       |
| Optimizing (1 sec. delay)  | 93.6               | 99.9       | 90.3                 | 99.0       |
| Optimizing (2 sec. delay)  | 93.4               | 96.2       | 84.9                 | 75.6       |
| Optimizing (3 sec. delay)  | 93.1               | 85.2       | 78.0                 | 45.5       |
| Optimizing (5 sec. delay)  | 89.4               | 61.4       | 59.0                 | 5.4        |
| Optimizing (7 sec. delay)  | 85.2               | 33.6       |                      |            |
| Optimizing (10 sec. delay) | 79.4               | 12.3       |                      |            |
| Optimizing (15 sec. delay) | 73.5               | 2.3        |                      |            |

We have investigated empirically this tradeoff between schedule quality and reaction time using a simulation of an online scheduling problem. To do this, we have implemented two different schedulers for this problem, one using combinatorial optimization and the other employing a dispatch rule. We have varied the simulated reaction time (scheduling delay) for the optimizing algorithm (while assuming that the dispatch rule reacts virtually instantaneously). To test the algorithms under different conditions, we have created datasets with different statistical properties. We have evaluated the scheduling performance under different combinations of scheduling algorithm, dataset, and reaction time. By doing so, we have verified experimentally that indeed there does exist a tradeoff between reaction time and schedule quality. Furthermore, less predictable, more disruptive data tilts the tradeoff further towards reaction time and enables the dispatch rule to outperform the optimizing scheduler with only a short scheduling delay.

There is much work that remains to be done in this area. For one, the experimental results are just preliminary. There are multiple dimensions of unpredictability and disruptiveness, and exploring these dimensions would require more variety than provided by the two datasets we used. Secondly, it would be highly desirable to have a better theoretical understanding of how the scheduling data affects the tradeoff between schedule quality and reaction time. Ideally, there exists some well-defined measure of unpredictability and how that affects the choice of scheduling algorithm. As a purely speculative example, consider the possibility of some defined fundamental time constant that says how long on average it takes before the scheduling problem changes by a certain fraction, roughly equivalent to the concept of half-life.

While there remains more to be done, we have taken some significant first steps by empirically demonstrating the dependence of the tradeoff between schedule quality and reaction time on the scheduling data, and by provid-

ing an approach for an experimental investigation of this tradeoff.

## Acknowledgements

This work was funded by DARPA UltraLog contract number MDA972-01-C-0025.

## References

- [1] BBN Technologies: 2004, ‘Vishnu Reconfigurable Scheduler Home Page’. <http://vishnu.bbn.com>.
- [2] Bent, R. and P. V. Hentenryck: 2004, ‘Regrets Only. Online Stochastic Optimization under Time Constraints’. *Proc. 19th National Conference on Artificial Intelligence (AAAI’04)*.
- [3] Bertsekas, D. and D. C. non: 1999, ‘Rollout Algorithms for Stochastic Scheduling Problems’. *Journal of Heuristics* **5**(1), 89–108.
- [4] Burke, P. and P. Prosser: 1991, ‘A Distributed Asynchronous System for Predictive and Reactive Scheduling’. *International Journal for Artificial Intelligence in Engineering* **6**(3), 106–124.
- [5] Chang, H., R. Givan, and E. Chong: 2000, ‘Online Scheduling Via Sampling’. *Proc. Fifth International Conference on Artificial Intelligence Planning and Scheduling*. pp. 62–71.
- [6] Dertouzos, M. and A. Mok: 1989, ‘Multiprocessor Online Scheduling of Hard-Real-Time Tasks’. *IEEE Transactions on Software Engineering* **15**(12), 1497–1506.
- [7] Kunnathur, A., P. Sundararaghavan, and S. Sampath: 2004, ‘Dynamic rescheduling using a

- simulation-based expert system'. *Journal of Manufacturing Technology Management* **15**(2), 199–212.
- [8] Montana, D.: 2001, 'A Reconfigurable Optimizing Scheduler'. *Proc. Genetic and Evolutionary Computation Conference*. pp. 1159–1166.
  - [9] Montana, D., M. Brinn, S. Moore, and G. Bidwell: 1998, 'Genetic Algorithms for Complex, Real-Time Scheduling'. *Proc. IEEE International Conference on Systems, Man, and Cybernetics*. pp. 2213–2218.
  - [10] Montazeri, M. and L. V. Wassenhove: 1990, 'Analysis of Scheduling Rules for an FMS'. *International Journal of Production Research* **28**(4), 785–802.
  - [11] Panwalker, S. and W. Iskander: 1977, 'A Survey of Scheduling Rules'. *Operations Research* **25**(1), 45–61.
  - [12] Rana-Stevens, S., B. Lubin, and D. Montana: 2000, 'The Air Crew Scheduling System: The Design of a Real-world, Dynamic Genetic Scheduler'. In: *Genetic and Evolutionary Computation Conference Late Breaking Papers*. Morgan Kaufmann.
  - [13] Rangsaritratsamee, R., W. Ferrell, and M. Kurz: 2004, 'Dynamic rescheduling that simultaneously considers efficiency and stability'. *Computers and Industrial Engineering* **46**(1), 1–15.
  - [14] Smith, S.: 1994, 'OPIS: A Methodology and Architecture for Reactive Scheduling'. In: Zweben and Fox (eds.): *Intelligent Scheduling*. Morgan Kaufmann, pp. 29–66.
  - [15] Vieira, G., J. Hermann, and E. Lin: 2003, 'Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods'. *Journal of Scheduling* **6**(1), 39–62.

# **Real World Scheduling**

Full Papers

# JOB SHOP SCHEDULING WITH LOT-SIZING AND BATCHING IN AN UNCERTAIN REAL-WORLD ENVIRONMENT

Sanja Petrovic<sup>1</sup>, Carole Fayad<sup>1</sup> and Dobrila Petrovic<sup>2</sup>

<sup>1</sup>*School of Computer Science & IT, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK, {sxp,cxf}@cs.nott.ac.uk}*

<sup>2</sup>*School of Mathematical and Information Sciences, Coventry University, Priory Street, Coventry CV1 5FB, UK, {D.Petrovic@coventry.ac.uk}*

**Abstract:** This paper proposes a novel algorithm for a real-world job shop-scheduling problem, where both lot-sizing and batching processes are considered. A fuzzy rule-based system is developed for determining lot sizes, where the input variables are workload on the shop floor, size of the job and its urgency while the output is the size of the lots. Both input and output variables are modelled as linguistic variables with imprecise values represented by using fuzzy sets. A fuzzy multi-objective genetic algorithm is developed to generate schedules of jobs whose processing times and due dates are imprecise and modelled by using fuzzy sets. A genetic algorithm takes into consideration the determined size of lots for jobs, and considers batching together jobs of similar characteristics in order to reduce the required setup time. The objectives considered are to minimize average tardiness, number of tardy jobs, setup times, idle times of machines and throughput times of jobs. The developed algorithm is tested on real-world data obtained from a printing company.

**Key words:** Job shop scheduling, lot-sizing, batching, fuzzy inference system, fuzzy multi-objective genetic algorithm, real-world application

## 1. INTRODUCTION

A job shop scheduling problem is an optimisation problem in which  $N$  jobs have to be allocated to  $M$  machines. Each job is assigned a predefined order of processing on the machines. Job shop scheduling problems attracted a considerable interest of the scheduling community and consequently the literature on this topic is rather rich (Pinedo, 2002; Yamada and Nakano, 1997, to mention just a few). In spite of this, there is still a gap between the academic research on job shop scheduling and real-world problems.

In this paper, we consider two issues within job shop scheduling and these are batching and lot-sizing that are important in many real-world scheduling problems. Batching refers to the decision of scheduling similar jobs contiguously, while lot-sizing deals with the decision on when and how to split a job into lots (Potts, 1995; Yavuz, 1999).

Batching is of particular importance in manufacturing systems in which changing of type of product to be processed on a machine incurs a setup time/cost. A set of jobs is grouped into families on the basis of their production requirements. As a result, no setup is needed when jobs of the same family are scheduled

consecutively. However, scheduling of jobs that belong to different families, one after the other, incurs setup time/costs.

The size of a batch appears to be a sensitive issue. Large batches improve the efficiency of the production line as they save time and also increase the machine utilization. On the other hand, a batch may group jobs of different priorities, and consequently, jobs of high priorities can be processed in different batches. Therefore, a customer service may be improved by having smaller batches. The recent survey of Potts and Kovalyov (2000) gives the extensive literature on models that integrate scheduling with batching decisions. They review models of batch scheduling developed for all classical machine settings. Agnetis et al. (2004) deal with the problem of finding a schedule that minimises the number of setups when jobs are subject to precedence constraints.

The concept of lot-sizing is introduced to partially satisfy customer's demand and to produce and deliver, in the first instance, *enough* amount of a product so that the customer is not out of stock of requested product while awaiting the whole job to be completed. The remaining part of the job will be delivered at some later date. In job-shop scheduling applications, the creation of lots permits the overlapping of different operations of the same job on parallel machines and may therefore also reduce throughput time (Potts, 1995). The sizes of lots have to be carefully chosen in order to avoid a schedule with a large number of lots of very small size that need large setup time on machines. The initial research on lot-sizing was related to the Economic Order Quantity (EOQ), where demand for a product is assumed to be stationary, i.e., an order for a product is continuously placed with a constant rate of demand within an infinite planning horizon. Economic Lot Scheduling problem (ELSP) takes into consideration capacity constraints (for example, machines capacity, inventory space capacity, etc.). A quite major step was made by introducing dynamic demand conditions. The Wagner-Whitin problem was introduced, which subdivided the finite planning horizon into discrete periods (Drexel and Kimms, 1997). Current approaches to lot-sizing alternate between capacitated (Sarker and Newton, 2002), discrete (Ouenniche and Bector, 1998) or a combination of these two approaches (Drexel and Kimms, 1997). Karacapilidis et al. (2000) introduced a fuzzy methodology for lot-sizing of multiple products in a production system, where demand for a product is expressed by linguistic terms.

The research described in this paper investigates batching and lot-sizing problems in a printing company Sherwood Press, in Nottingham, UK. Decisions regarding batching and lot-sizing are made in the presence of uncertainty. Processing times of jobs are uncertain due to both machines and human factors, while due dates are also *flexible* and allow the decision maker (scheduler) to express his/her attitude toward the tardiness of jobs. Processing times and due dates of the jobs are modelled by fuzzy sets. Through the series of interviews with Sherwood Press staff, variables identified to be important for lot-sizing are workload on the shop floor, job size and job urgency. In order to mimic the way that the scheduler decides on the lots and lot-sizing, fuzzy IF-THEN rules are introduced which derive conclusions based on imprecise premises. A genetic algorithm (GA) is developed which takes into consideration uncertain processing times and due dates of jobs and the decision on lot-sizing of jobs derived using the fuzzy rules. A number of objectives are used simultaneously during the GA search to measure the quality of the schedules. These objectives are average tardiness, number of tardy jobs, total setup time, total idle time of machines and total throughput time of jobs.

The paper is organized as follows. Section 2 introduces a job shop problem and the real-world instance of the problem present at Sherwood Press. Section 3 describes fuzzy IF-THEN rules developed for lot-sizing, while a fuzzy GA for job shop scheduling is introduced in Section 4. Section 5 discusses the results obtained on real-world data obtained from Sherwood Press, followed by conclusions.

## 2. PROBLEM STATEMENT

The notation used in the problem statement and throughout the paper is as follows.

$N$                                       total number of jobs

|           |                                                                                                                                                 |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| $J_j$     | job, $j=1, \dots, N$                                                                                                                            |
| $M$       | total number of machines                                                                                                                        |
| $M_i$     | machine, $i=1, \dots, M$                                                                                                                        |
| $F$       | total number of families of jobs                                                                                                                |
| $F_f$     | family of jobs, $f=1, \dots, F$                                                                                                                 |
| $H$       | total number of unit time periods of the planning horizon                                                                                       |
| $t$       | period in the planning horizon, $t=1, \dots, H$                                                                                                 |
| $q_j$     | quantity of identical items in job $J_j$ , $j=1, \dots, N$                                                                                      |
| $r_j$     | release date of job $J_j$ , $j=1, \dots, N$ , which denotes when a job can start its processing                                                 |
| $s_j$     | start date of job $J_j$ , $j=1, \dots, N$ , which denotes when a job has started its processing                                                 |
| $d_j$     | due date of job $J_j$ , $j=1, \dots, N$                                                                                                         |
| $C_j$     | completion date of job $J_j$ , $j=1, \dots, N$                                                                                                  |
| $C_{max}$ | makespan $C_{max} = \max(C_1, \dots, C_j, \dots, C_n)$                                                                                          |
| $T_j$     | tardiness of job $J_j$ , $j=1, \dots, N$<br>$T_j = \max(C_j - d_j, 0)$                                                                          |
| $w_j$     | importance of job $J_j$ , $j=1, \dots, N$                                                                                                       |
| $(i,j)$   | operation of job $J_j$ , $j=1, \dots, N$ processed on machine $M_i$ , $i=1, \dots, M$                                                           |
| $p_{ij}$  | processing time of operation $(i,j)$ given in unit time<br>periods, $i=1, \dots, M$ , $j=1, \dots, N$                                           |
| $L_j$     | number of lots of job $J_j$                                                                                                                     |
| $(i,j,l)$ | an operation $(i,j)$ of lot $l$ , $i=1, \dots, M$ , $j=1, \dots, N$ and $l=1, \dots, L_j$                                                       |
| $q_{ijl}$ | number of identical items of an operation $(i,j,l)$ ,<br>$i=1, \dots, M$ , $j=1, \dots, N$ and $l=1, \dots, L_j$                                |
| $p_{ijl}$ | processing time of an operation $(i,j,l)$ , $i=1, \dots, M$ ,<br>$j=1, \dots, N$ and $l=1, \dots, L_j$ , $p_{ijl} = \frac{p_{ij}}{q_j} q_{ijl}$ |
| $C_{ij}$  | completion time of an operation $(i,j)$ , $i=1, \dots, M$ ,<br>$j=1, \dots, N$                                                                  |

The job shop scheduling problem considered in this research consists of  $N$  jobs  $\{J_1, \dots, J_j, \dots, J_N\}$  with given release and due dates  $r_j, d_j$ ,  $j=1, \dots, N$ , respectively, that have to be scheduled on a set of  $M$  machines  $\{M_1, \dots, M_i, \dots, M_M\}$ . Each job consists of a chain of operations on which precedence constraints are imposed. Each operation of job  $J_j$  processed on machine  $M_i$  is represented by an ordered pair  $(i,j)$ , while

its processing time is denoted by  $p_{ij}$ . Each job belongs to a certain family. Families of jobs are determined on the basis of the colour requirements of jobs.

The scheduling problem is formulated as follows. Find a non-preemptive sequence of operations of  $N$  jobs on each of  $M$  machines taking into consideration the following objectives:

(I) to minimise the average tardiness of schedule  $s$

$$AT(s) = \frac{1}{n} \sum_{j=1}^N w_j T_j \quad (1)$$

(II) to minimise the number of tardy jobs in schedule  $s$

$$NT(s) = \sum_{j=1}^N u_j, \text{ where } u_j = \begin{cases} 1 & \text{if } T_j > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

(III) to minimise the total setup time in schedule  $s$

$$ST(s) = \sum_{t=1}^H \sum_{f=1}^F a X_{ft}, \quad (3)$$

where  $X_{ft} = \begin{cases} 1 & \text{if there is a need for a setup for a family of jobs at period } t \\ 0 & \text{otherwise} \end{cases}$

$X_{ft}$  is referred to as the changeover coefficient and parameter  $a$  has the value of 20 minutes, which is the time needed to make a machine ready for a changeover of the job families.

(IV) to minimise the total idle time of machines in schedule  $s$ , where the idle time is defined as time within the planning horizon during which the machines are used neither for processing of jobs nor for setup

$$IT(s) = H.M - \left( \sum_{t=1}^H \sum_{f=1}^F a X_{ft} \right) - \left( \sum_{i=1}^M \sum_{j=1}^N \sum_{l=1}^{L_j} p_{ijl} \right) \quad (4)$$

(V) to minimise the total throughput time defined as the total time that jobs spend on the shop floor in a schedule  $s$

$$TT(s) = \sum_{j=1}^N (C_j - s_j); \quad j = 1, \dots, N \quad (5)$$

## 2.1 Real-world Job Shop Problem

A job shop problem present in Sherwood Press involves a number of machines ( $M=18$ ), which are grouped into 7 work centres: Printing, Cutting, Folding, Card-inserting, Embossing and Debossing, Gathering, Finishing (Stitching, Trimming and Packaging). Each job (a printing material) has a predetermined order of processing on machines in some or all work centres. Processing times of jobs are uncertain and represented by fuzzy sets. The definition of fuzzy sets is given in the Appendix.

The *estimation* of processing time of each operation is obtained taking into consideration the nature of the machine in use. While some machines are automated and can be operated at different speeds, others are staff-operated and therefore the processing times are staff-dependent. Uncertain processing times  $\tilde{p}_{ij}$  are

modelled by triangular membership functions represented by a triplet  $(p_{ij}^1, p_{ij}^2, p_{ij}^3)$ , where  $p_{ij}^1$  and  $p_{ij}^3$  are lower and upper bounds of the processing time, while  $p_{ij}^2$  is a so-called a modal point (Klir, 1998). An example of fuzzy processing time is shown in Figure 1.

Following the scheduling practice in Sherwood Press, we classify jobs into three groups on the basis of their priorities and the corresponding tolerance of their tardiness: *High*, *Medium* and *Low*. A job of *High* priority is expected to be processed on time with no delay, and therefore there is a crisp due date defined for it. A job of *Medium* priority has a due date that can be extended with two additional days, whereas a job of *Low* priority is given a tolerance of delay of a maximum of one week on top of the originally set due date. The imprecise due dates  $\tilde{d}_j, j = 1, \dots, N$ , are represented by trapezoidal fuzzy sets illustrated in Figure 2.

### 3. FUZZY LOT SIZING

In order to deliver at least part of the job ordered on time, jobs are split into lots. Three variables are identified to be important for the decision on lot-sizing: the *workload* on the shop floor, *size of the job* and *urgency of the job* with respect to its due date. Fuzzy IF-THEN rules are defined to decide on the lot-sizing using imprecise values of these three variables. *Workload* takes into consideration the number of items to be processed on the shop floor and is described by two linguistic terms: *Not Large* and *Large*. *Urgency of the job* is calculated as the difference between its due date and release date and is described by three linguistic terms: *High*, *Medium* and *Low*. *Very Small*, *Small*, *Medium* and *Large* are the linguistic terms used to describe *size of the job*. These membership functions are defined in collaboration with Sherwood Press following their policy for lot-sizing. They are given in Figure 3.

Fuzzy IF-THEN rules are defined based on the policy of Sherwood Press regarding lot-sizing. In order to keep the number of required setups as low as possible, jobs are split into maximum two lots. Therefore, the purpose of the fuzzy IF-THEN rules is to decide whether to split a job into lots and on the size of each lot which is given as percentage of the size of the job.

It would be very difficult to define all 24 fuzzy rules, which determine the size of lots for all possible values of the three variables (*workload*, *size of job*, *urgency of job*) in the premises of the rules. To overcome this difficulty, we define fuzzy rules for each of the three variables independently and apply them in sequence (Petrovic, 1991). A conclusion variable *change of lot* is introduced in the fuzzy rules, which indicates a necessary change of the current size of the lots. It is described by linguistic values *Large Negative*, *Medium Negative*, *Small*, *Medium Positive*, *Large Positive*. The membership functions of the corresponding fuzzy sets are given in Figure 4.

The starting point in the lot-sizing decision is to split each job into two lots of equal size, i.e., 50% of the job size. A set of fuzzy rules is defined which considers *workload*, *size of jobs* and *urgency of jobs* independently in order to derive a *change of lot*. The final size of the two lots is obtained as an aggregated result of firing of all the rules. Fuzzy rules are given in Figure 5.

The rules attempt to reflect a policy of the company. For example, if the *workload* is *Large*, then the idea is to have the first lot of smaller size and to *postpone* the processing of the major part of the job. Similar holds true for *Large* jobs. Opposite to that, smaller jobs are processed as much as possible in the first lots. On the other hand, if a job is of *High urgency* it should be processed completely as soon as possible, i.e., the first lot should be rather large, while jobs of *Low urgency* will be left mostly for processing in the second lots.

A fuzzy rule based system developed for lot-sizing comprises four components (Negnevitsky, 2002):

1. A fuzzifier takes crisp input values and determines the degrees to which they belong to fuzzy sets in the premises of the rules.



2. A fuzzy rule base given in Figure 5 contains rules with one imprecise variable in the premises of the rules (*workload, size of job or job urgency*) and imprecise conclusion (*change of lot*).

Generally, fuzzy rule  $R_s$  with one premise and one conclusion has the following form:

$$R_s: \quad \text{IF } x \text{ is } A_s \text{ THEN } y \text{ is } B_s$$

where  $x$  and  $y$  are imprecise variables,  $A_s$  and  $B_s$ ,  $s = 1, \dots, S$  are fuzzy sets,  $S$  is the total number of rules.

3. A fuzzy inference engine derives imprecise conclusions based on imprecise premises. The truth-value of a rule premise is equal to the value of the membership function of its premise  $\mu_{A_s}(x_0)$  for a given crisp input  $x_0$ . The truth-value of the conclusion is determined using the *cutting* (or *truncation*) method, where the membership function of the fuzzy set in the conclusion of the rule is cut at the level of the obtained truth-value  $\mu_{A_s}(x_0)$  of the corresponding premise. The obtained fuzzy set is denoted by  $\mu_{A_s \rightarrow B_s(x_0)}(y)$  and has the following membership function:

$$\mu_{A_s \rightarrow B_s(x_0)}(y) = \min\{\mu_{A_s}(x_0), \mu_{B_s}(y)\} \quad (6)$$

A final conclusion  $R$  is a fuzzy set obtained by aggregating conclusions of all the rules that have fired. Maximum operator is used for aggregation as follows:

$$\mu_R(y) = \max\{\mu_{A_1 \rightarrow B_1(x_0)}(y), \dots, \mu_{A_S \rightarrow B_S(x_0)}(y)\} \quad (7)$$

4. A defuzzifier maps a final fuzzy set obtained by the fuzzy inference engine into a crisp value  $\bar{y}$ , which represents the fuzzy set “the most suitable” and “correctly”. Our system uses a centre of gravity which is often applied defuzzifier:

$$\bar{y} = \frac{\sum_{s=1}^S (y * \mu_R(y))}{\sum_{s=1}^S \mu_R(y)} \quad (8)$$

#### 4. A FUZZY GENETIC ALGORITHM FOR JOB SHOP SCHEDULING

A genetic algorithm (GA) is an iterative search procedure widely used for solving combinatorial optimisation problems, motivated by biological systems and natural genetics (Reeves, 1995). The main characteristics of the GA developed for job shop scheduling with batching and lot-sizing are described below (further details on the GA are given in Fayad and Petrovic (2005)).

Each chromosome consists of two sub-chromosomes of length  $M$  (number of machines), named *machines sub-chromosome* and *dispatching rules sub-chromosome*. The genes of the first sub-chromosome contain machines, while genes of the second sub-chromosome contain the dispatching rules to be used for sequencing operations on the corresponding machines. The machine sub-chromosome is filled in by randomly choosing machines  $M_i$ ,  $i=1, \dots, M$ . *Dispatching rules sub-chromosome* is initialised by choosing randomly one among the following six rules: *Early Due Date First*, *Shortest Processing Time First*, *Longest Processing Time First*, *Longest Remaining Processing Time First*, *Highest Priority First*, *Same Family of Jobs Together*. While the first four dispatching rules are well-established and widely used in the literature on job shop scheduling (Pinedo, 2002), the last two are tailored to this application with the aim to reduce throughput time of jobs of higher priority and to schedule, whenever possible, jobs of the same families contiguously.

The fitness function is used to evaluate the quality of a given schedule driving the search towards *promising* areas. In our problem, objectives are measured in different units i.e., they are incommensurable. For example, average tardiness of jobs are given in time units, while the number of tardy jobs takes an integer value from the interval  $[0, N]$ . However, the objective values have to be used simultaneously to assess the quality of schedules. Satisfaction grades are introduced for each objective to reflect the decision maker preferences with respect to the achieved values of the objectives. Values of the objectives are mapped

into satisfaction grades, which take values from the interval  $[0, 1]$ , where 0 represents full dissatisfaction and 1 full satisfaction with the achieved objective value. The satisfaction grades with the achieved values of all the objectives are combined in an overall satisfaction grade.

Fuzzy processing times of job operations imply fuzzy completion times of jobs. Before we can assess the satisfaction grades of objectives that are completion and due date related (given in (I) and (II)), we need to establish a way to compare a fuzzy completion time of a job with its fuzzy due date and therefore calculate the likelihood that a job is tardy. We use the method of area of intersection introduced by Sakawa (2000) which measures the intersection between the two fuzzy values  $\tilde{C}_j$  and  $\tilde{d}_j$  (Figure 6). The satisfaction grade of a fuzzy completion time of job  $J_j$  is defined in the following way:

$$S_T(\tilde{C}_j) = (\text{area } \tilde{C}_j \cap \tilde{d}_j) / (\text{area } \tilde{C}_j) \quad (9)$$

The objectives (I) to (V) are transformed into the objectives to maximize the corresponding satisfaction grades as follows.

(I) Satisfaction grade of Average Tardiness  $S_{AT}$

$$S_{AT} = \frac{1}{N} \sum_{j=1}^N S_T(\tilde{C}_j) \quad (10)$$

(II) Satisfaction grade of Number of Tardy Jobs  $S_{NT}$  is presented in Figure 7

$$S_{NT} = \begin{cases} 1 & \text{if } nTardy = 0 \\ (Max_{NT} - nTardy) / Max_{NT} & \text{if } 0 < nTardy < Max_{NT} \\ 0 & \text{if } nTardy \geq Max_{NT} \end{cases} \quad (11)$$

where  $nTardy$  is the number of tardy jobs and  $Max_{NT}$  is equal to 15% of the total number of jobs and presents a preference of the scheduler at Sherwood Press; there is full dissatisfaction when  $nTardy \geq Max_{NT}$ , and full satisfaction when  $nTardy = 0$ , while satisfaction degrees are linearly decreasing when  $nTardy$  increases from 0 to  $Max_{NT}$ .

Parameter  $\lambda$  is introduced to define when job  $J_j, j=1, \dots, N$  is considered to be tardy. It is the case when

$$S_T(\tilde{C}_j) \leq \lambda, 0 \leq \lambda \leq 1. \text{ In our problem } \lambda \text{ is set empirically to be } 0.6.$$

(III) Satisfaction grade of Total Setup Time  $S_{ST}$  (Figure 8)

$$S_{ST} = \begin{cases} 1 & \text{if } Total_{ST} = 0 \\ (Max_{ST} - Total_{ST}) / Max_{ST} & \text{if } 0 < Total_{ST} < Max_{ST} \\ 0 & \text{if } Total_{ST} > Max_{ST} \end{cases} \quad (12)$$

where  $Total_{ST} = \sum_{t=1}^H \sum_{f=1}^F aX_{ft}$ , where a setup  $X_{ft}$  is applied whenever two jobs of different families are scheduled contiguously. We assume that a setup requires  $a = 20$  minutes each time it is applied.

A maximum setup time  $Max_{ST}$  is needed when all consecutive jobs belong to different families, i.e.,  $Max_{ST} = a \cdot \sum_{i=1}^M \sum_{j=1}^N (i, j)$ , where  $\sum_{i=1}^M \sum_{j=1}^N (i, j)$  is the total number of operations.

(IV) Satisfaction Grade of Total Idle times  $S_{IT}$  (Figure 9)

$$S_{IT} = \begin{cases} 1 & \text{if } Total_{IT} = 0 \\ (Max_{IT} - Total_{IT}) / Max_{IT} & \text{if } 0 < Total_{IT} < Max_{IT} \\ 0 & \text{if } Total_{IT} > Max_{IT} \end{cases} \quad (13)$$

The Total Idle Time is

$$Total_{IT} = \sum_{i=1}^M \left( (C_{ia} - s_{ib}) - \sum_{j=1}^N p_{ij} \right) - \sum_{t=1}^H \sum_{f=1}^F aX_{ft} \quad \text{where } (i,a) \text{ and } (i,b) \text{ are the last operation finished and}$$

the first operation started on machine  $M_i$  within planning horizon  $H$ , respectively.

$\sum_{j=1}^N p_{ij}$  is the total processing times of all jobs on machine  $M_i$

$Max_{IT}$  is the total time of machines being in use  $Max_{IT} = \sum_{i=1}^M (C_{ia} - s_{ib})$

(V) Satisfaction Grade of Total Throughput Time  $S_{TT}$

$$S_{TT} = \begin{cases} 1 & \text{if } Total_{TT} = 0 \\ (Max_{TT} - Total_{TT}) / Max_{TT} & \text{if } 0 < Total_{TT} < Max_{TT} \\ 0 & \text{if } Total_{TT} > Max_{TT} \end{cases} \quad (14)$$

where the Total Throughput Time shows the time the jobs are on the shop floor,  $Total_{TT} = \sum_{j=1}^N (C_j - s_j)$

Max Throughput Time is  $Max_{TT} = \sum_{j=1}^N (C_{max} - s_j)$

Once the satisfaction grades with the values of all the objectives are determined, the average aggregation operator is used to combine them into one satisfaction grade  $F$ :

$$F = (S_{AT} + S_{NT} + S_{ST} + S_{IT} + S_{TT}) / 5 \quad (15)$$

## 5. PERFORMANCE OF THE FUZZY GA ON THE REAL-WORLD DATA

Data obtained from Sherwood Press were used to test the performance of the developed fuzzy GA with lot-sizing and batching. The input data for each job are *JobID*, the order of machines that the job requires its processing on, uncertain processing time on each of the machines, required quantity of the items, family of the job, release date, due date and the priority of the job. The planning horizon is 4 weeks. Based on the release date and due date of the job, each job is assigned its urgency. The workload is calculated taking into consideration the required production of items. Jobs of the same family are grouped together into batches whenever possible. Batching is performed only on the printing machines.

Two groups of tests were performed with the aim (a) to evaluate the performance of the IF-THEN fuzzy rules for lot-sizing and (b) to evaluate the performance of the fuzzy GA which takes as input data the number of lots for each job (none or 2) and the size of each lot if the jobs are split into lots. The algorithm was implemented using Visual C++ in a Windows XP environment, and tests were run on a PC Pentium 2 GHz with 512 MB of RAM.

5.1 Testing of fuzzy rules for lot-sizing

A number of jobs with specified *size* and *urgency*, given in Table 1, were released. The *workload* of the shop floor was 5120125 items which required processing. The data in column *change of lot* shows how much the initial lot size of 50% has to be changed to reflect the *size* and the *urgency of jobs*. For example, job #7 of *size* 460, considered as *Very Small* (with membership degree 1) and of *urgency* 5, considered as *High* (with membership degree 0.25), will not be split into lots and therefore the total size of the first lot is 100% of the job. Also, two jobs (#1 and #2) of similar *urgency* will be split up differently due to a difference in their sizes. Job of the larger size (#1) will be split into lots of 37% and 63% of the size of the job, while in the case of job #2 of the smaller size, the larger part of the job (63%) will be processed in the first lot and the remaining part (37%) in the second lot. Considering jobs #3 and #6, which are of the same *urgency*, job #3 is larger and therefore its first lot is smaller (47%) than the second lot (53%). Job #6 is of the smaller size and the major part will be processed in the first lot (61%).

5.2 Testing of fuzzy genetic algorithm

The developed fuzzy GA was run with the parameters given in Table 2. The algorithm was run 5 times and the results obtained using real-world data for one month planning horizon discretised homogeneously into one hour unit time periods, are given in Table 3. This specific month was chosen because it was considered to be rather busy with 158 jobs and a *workload* of 5120125 required items. In order to study the effects of lot-sizing on the performance of the schedule, the algorithm was also run with no lot-sizing.

In the calculations of due-date oriented objective values achieved (*AT* and *NT*) in the schedules with lots we take into consideration the completion of jobs in the second lot (if the job was split in two lots). Therefore results presented in Table 3 need to be interpreted carefully having in mind that the splitting of jobs into lots leads to higher number of operations but gives *better* opportunities for batching. It can be observed that the average tardiness achieved in both cases, with and without lots, have similar values, while the number of tardy jobs is slightly lower in the schedules with lots (i.e., the average and the best satisfaction grades are higher). However, this is a consequence of the approach taken to calculate the tardiness of the schedule with lots when the second lot is taken into consideration only. In order to investigate the effect of splitting jobs into lots on tardiness, we also record the number of tardy jobs taking into consideration completion of jobs in the first lots. Table 4 shows the number of tardy jobs (out of 158) taking into consideration two lots and the first lot, respectively. Of course, the second lots determine the ultimate quality of the schedule but the smaller number of tardy jobs in the first lots may increase the satisfaction grade of the customer.

Contrary to the expectation, the obtained values of the total setup time are lower in the schedules with lots, although these schedules consist of higher number of operations. The possible explanation for that lies in batching. Namely, in schedules with lots, jobs are split up into operations of smaller sizes in terms of the number of items to be processed, i.e., there is a higher number of operations to be scheduled than in the schedules without lots. It leads to the smaller number of batches and higher average size of batches in the schedules with lots than in the schedules without lots. The total number of batches, the total number of operations to be scheduled and average size of batches on the 3 printing machines are given in Table 5 (there are 3 printing machines in Sherwood that batching can take place on).

The developed GA uses 6 different dispatching rules to sequence job operations on each of the machines. These dispatching rules are first chosen with the same probability. Further experiments are performed with assigning a higher probability to some of the rules. For example, in order to increase the satisfaction grade

for the total setup time, the probability of applying the dispatching rule *Same Family of Jobs Together* is increased. As expected, the total setup time was improved (the satisfaction grade was increased from 0.327 to 0.6, but at an expense of the number of tardy jobs which increases from 11 to 73 (out of a total of 158 jobs). Similar experiments can be performed with other dispatching rules.

In the initial algorithm, the maximum number of lots was set to be 2. The question arises whether three lots can improve the performance of a schedule. The algorithm was applied as follows. The fuzzy IF-THEN rules are firstly applied to determine whether to split the job into lots and to decide on the size of the two lots. Then, the second lot was considered as a new problem and the same set of fuzzy rules was used to decide whether to split it into lots and to determine the sizes of the two additional lots. The obtained values of the objective functions are given in Table 6. We can conclude that they are similar to the values obtained with 2 lots.

## 6. CONCLUSION

The paper considers a real-world job shop problem that is present in a printing company. Processing times of jobs and their due dates are imprecise and modelled by fuzzy sets. Fuzzy rules, which reflect the scheduling policy of the printing company, are defined to determine lot sizes. A fuzzy multi-objective genetic algorithm is developed to generate schedules of jobs. The objectives considered are to minimize average tardiness, number of tardy jobs, setup times, idle times of machines and throughput times of jobs. Fuzzy sets are used to represent satisfaction grades for the objectives taking into consideration the preferences of the scheduler on the shop floor. A genetic algorithm is developed to search for a schedule with maximum satisfaction grades for the objectives. The algorithm uses lot sizing of jobs determined by the fuzzy rule-based system. In addition, the batching of jobs with similar characteristics is taken into consideration, which leads to reduce the total setup time of machines. The results obtained on real-world data from a printing company are given and analyzed.

Our future work will be focused on additional issues regarding the lot sizing and batching. We will consider scheduling of lots on the alternative parallel machines on the shop floor at the same time. Batching of jobs is important in real-world problems because it reduces setup time of the machines. In the current algorithm it is handled by one dispatching rule, which groups jobs of the same family together in one batch. We will investigate whether a new objective, which maximises number of batches can improve the performance of the schedule.

## ACKNOWLEDGMENTS

The authors would like to thank the Engineering and Physics Science Research Council (EPSRC), UK, for supporting this research (Grant No. GR/R95319/01). We also acknowledge the support of the industrial collaborator Sherwood Press Ltd, Nottingham.

## REFERENCES

- Agnetis A., Alfieri A. and Nicosia G. (2004), A Heuristic Approach to Batching and Scheduling a Single Machine to Minimise Setup Costs, *Computers & Industrial Engineering*, 46, 2, 793-802.
- Drexel A. and Kimms A. (1997), Lot Sizing and Scheduling – Survey and Extensions, *European Journal of Operational Research*, 99: 221-235.
- Fayad C. and Petrovic S. (2005), A Genetic Algorithm for the Real World Job-Shop Scheduling, accepted for publication in the *Proceedings of The International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-2005*, June 22-25, 2005, Bari, Italy, IEA/AIE 2005, Lecture Notes of Artificial Intelligence (LNAI).
- Karapapilidis N., Pappis C. and Adamopoulos G. (2000), Fuzzy Set Approaches to Lot Sizing, in Slowinski R. and Hapke M. (Eds), *Scheduling Under Fuzziness*, Physica-Verlag, Heidelberg, ISBN 3-7908-1249-8, pp 291-304.

- Klir G. and Folger T. (1998), *Fuzzy Sets, Uncertainty and Information*, Prentice Hall, New Jersey.
- Negnevitsky, M. (2002), *Artificial Intelligence: A Guide to Intelligent Systems*, Addison Wesley, ISBN: 0201711591.
- Ouenniche J. and Boctor F. (1998), Sequencing, Lot Sizing and Scheduling of Several products in Job Shops: The Common Cycle Approach, *International Journal of Production Research*, 36, 4, 1125-1140.
- Petrovic D. (1991), Decision Support for Improving Systems Reliability by Redundancy, *European Journal of Operational Research*, 55: 357-367.
- Pinedo M. (2002), *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, New Jersey, Second Edition, ISBN: 0-13-028138-7.
- Potts C. N. and Wassenhove L. N. Van (1995), Integrating Scheduling with Batching and Lot-Sizing: a Review of Algorithms and Complexity, *Journal of Operational Research Society*, 43, 5, 395-406.
- Potts C. and Kovalyov M. (2000), Scheduling with Batching: A Review, *European Journal of Operational Research*, 120: 228-249.
- Reeves C. (1995), Genetic Algorithms, in V.J. Rayward-Smith (Eds), *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill International, UK, ISBN 0-07-709239-2, pp 151-196.
- Sakawa, M., and Kubota, R. (2000), Fuzzy Programming for Multiobjective Job Shop Scheduling with Fuzzy Processing Time and Fuzzy Duedate through Genetic Algorithms, *European Journal of Operational Research*, 120, 2, 393-407.
- Sarker R. and Newton C (2002), A Genetic Algorithm for Solving Economic Lot Size Scheduling Problem, *Computers & Industrial Engineering*, 42: 189-198.
- Yamada T. and Nakano R. (1997), Job Shop Scheduling, in Zalzal A. and Fleming P. (Eds), *Genetic Algorithms in Engineering Systems*, IEE Control Engineering Series 55, United Kingdom, ISBN 0-85296-902-3, pp 134-160.
- Yavuz S (1999), 'Scheduling with Batching and Lot-Sizing,' <http://www.ie.bilkent.edu.tr/~ie672/docs/trends.html>

## APPENDIX

Fuzzy set  $\tilde{A}$  is defined by a membership function  $\mu_{\tilde{A}}(x)$  which assigns to each object  $x$  in the universe of discourse  $X$ , a value that represents its grade of membership in the fuzzy set (Klir, 1998):

$$\mu_{\tilde{A}}(x): X \rightarrow [0,1]$$

A variety of shapes can be used for membership functions such as triangular, trapezoidal, bell curves and s-curves. Conventionally, the choice of the shape is subjective and allows the decision maker to express his/her preferences.

Table 1. Change of lot obtained using the fuzzy IF-THEN rules

| <i>JobID</i> | <i>size</i> | <i>urgency</i> | <i>change of lot</i> | <i>First lot</i> | <i>Second lot</i> |
|--------------|-------------|----------------|----------------------|------------------|-------------------|
| #1           | 70000       | 19             | -13%                 | 37%              | 63%               |
| #2           | 4180        | 18             | +13%                 | 63%              | 37%               |
| #3           | 70000       | 10             | -3%                  | 47%              | 53%               |
| #4           | 4500        | 3              | +29%                 | 79%              | 21%               |
| #5           | 596000      | 8              | -12%                 | 38%              | 62%               |
| #6           | 12000       | 10             | +11%                 | 61%              | 39%               |
| #7           | 460         | 5              | +50%                 | 100%             | 0%                |
| #8           | 84000       | 35             | -25%                 | 25%              | 75%               |
| #9           | 140000      | 12             | -15%                 | 35%              | 65%               |
| #10          | 60000       | 12             | -3%                  | 47%              | 53%               |

Table 2. Parameters of the GA

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| Population size          | 50                                                 |
| Length of the chromosome | 2 <i>M</i> , ( <i>M</i> is the number of machines) |
| Crossover probability    | 0.8                                                |
| Mutation probability     | 0.3                                                |
| Termination condition    | 500 iterations                                     |

Table 3. Average and best values of the aggregated satisfaction grades and satisfaction grades of the objectives, runs with and without lots

| Runs with lots    | <i>F</i> | <i>S<sub>AT</sub></i> | <i>S<sub>NT</sub></i> | <i>S<sub>ST</sub></i> | <i>S<sub>IT</sub></i> | <i>S<sub>TT</sub></i> |
|-------------------|----------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 1                 | 0.51     | 0.424                 | 0.588                 | 0.345                 | 0.368                 | 0.843                 |
| 2                 | 0.54     | 0.43                  | 0.651                 | 0.371                 | 0.372                 | 0.874                 |
| 3                 | 0.53     | 0.43                  | 0.652                 | 0.353                 | 0.369                 | 0.873                 |
| 4                 | 0.53     | 0.43                  | 0.652                 | 0.336                 | 0.368                 | 0.877                 |
| 5                 | 0.52     | 0.43                  | 0.683                 | 0.239                 | 0.373                 | 0.859                 |
| Average           | 0.53     | 0.43                  | 0.645                 | 0.329                 | 0.37                  | 0.865                 |
| Best              | 0.54     | 0.43                  | 0.652                 | 0.363                 | 0.37                  | 0.87                  |
| Runs without lots |          |                       |                       |                       |                       |                       |
| 1                 | 0.46     | 0.416                 | 0.525                 | 0.119                 | 0.384                 | 0.866                 |
| 2                 | 0.47     | 0.429                 | 0.588                 | 0.1                   | 0.379                 | 0.874                 |
| 3                 | 0.46     | 0.42                  | 0.556                 | 0.11                  | 0.38                  | 0.86                  |
| 4                 | 0.39     | 0.419                 | 0.557                 | 0.113                 | 0.379                 | 0.866                 |
| 5                 | 0.42     | 0.407                 | 0.367                 | 0.11                  | 0.375                 | 0.847                 |
| Average           | 0.44     | 0.418                 | 0.518                 | 0.110                 | 0.379                 | 0.862                 |
| Best              | 0.47     | 0.429                 | 0.588                 | 0.119                 | 0.384                 | 0.874                 |

Table4. Number of tardy jobs versus tardy first lots

| Instance | Tardy jobs | Tardy first lots |
|----------|------------|------------------|
| 1        | 14         | 10               |
| 2        | 12         | 9                |
| 3        | 8          | 5                |
| 4        | 11         | 9                |
| 5        | 14         | 8                |

Table 5. Batching with and without lots

| Run with | Number of batches | Number of operations | Batch Average Size |
|----------|-------------------|----------------------|--------------------|
| no lots  | 162               | 185                  | 1.13               |
| lots     | 179               | 369                  | 1.83               |

Table 6. Average and best values of the aggregated satisfaction grades and satisfaction grades of the objectives with 3 lots

| Run with 3 lots | $F$  | $S_{AT}$ | $S_{NT}$ | $S_{ST}$ | $S_{IT}$ | $S_{TT}$ |
|-----------------|------|----------|----------|----------|----------|----------|
| 1               | 0.55 | 0.43     | 0.65     | 0.44     | 0.38     | 0.87     |
| 2               | 0.54 | 0.43     | 0.65     | 0.42     | 0.39     | 0.87     |
| 3               | 0.52 | 0.43     | 0.65     | 0.28     | 0.39     | 0.86     |
| 4               | 0.52 | 0.43     | 0.65     | 0.28     | 0.38     | 0.86     |
| 5               | 0.50 | 0.42     | 0.55     | 0.34     | 0.38     | 0.85     |
| Average         | 0.53 | 0.428    | 0.63     | 0.352    | 0.384    | 0.862    |
| Best            | 0.55 | 0.43     | 0.65     | 0.44     | 0.38     | 0.87     |



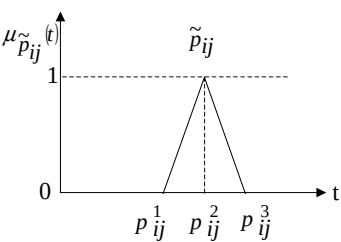


Figure 1. Fuzzy processing times

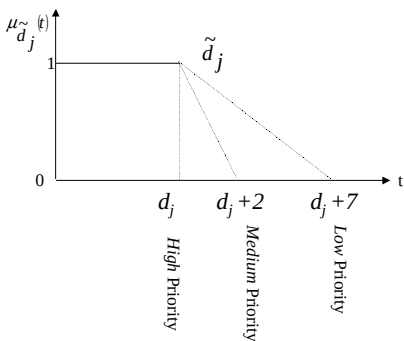


Figure 2. Fuzzy due dates

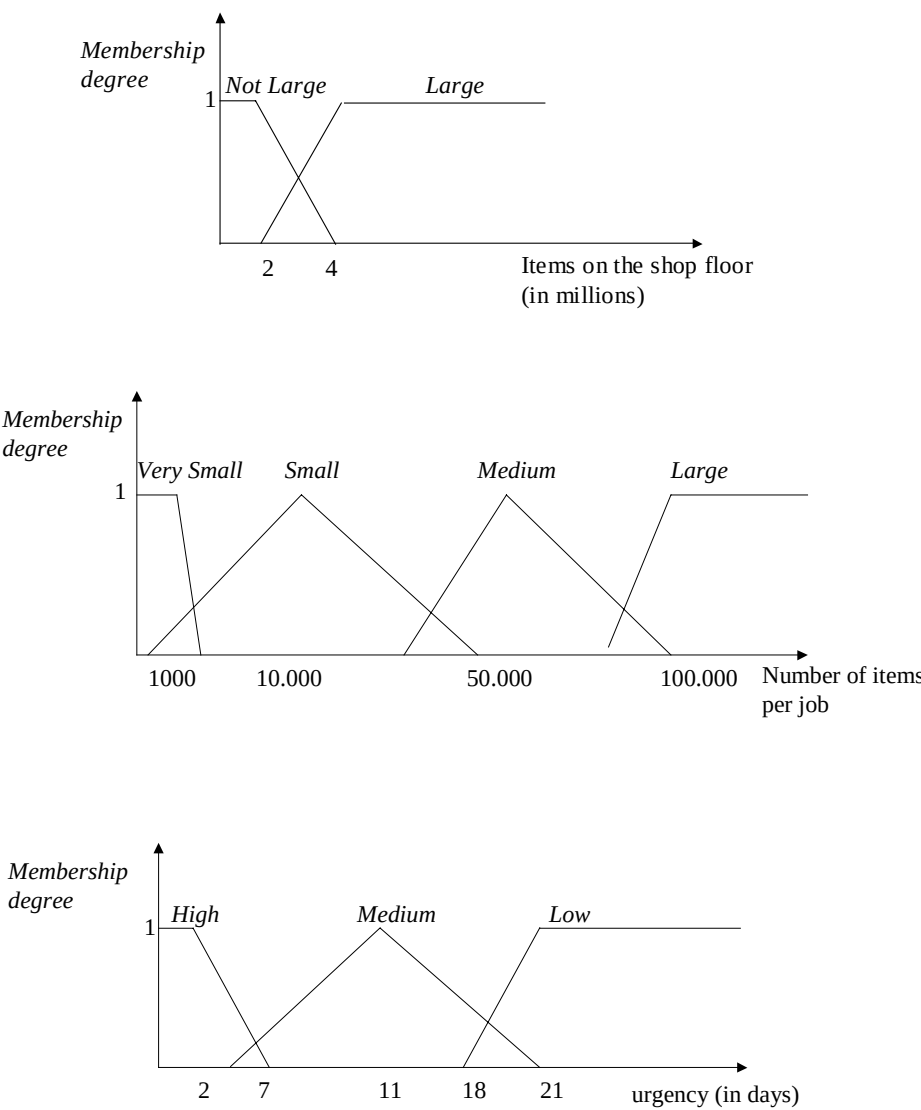


Figure 3. Linguistic terms for variables workload, urgency of the job, size of the job

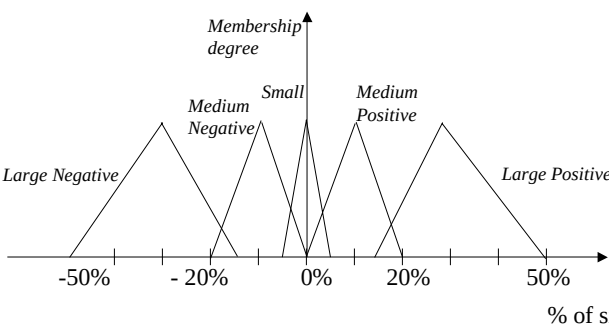


Figure 4. Membership function of the linguistic terms of the conclusion variable

1. IF workload is Not Large THEN change of lot is Small
  2. IF workload is Large THEN change of lot is Medium Negative
  3. IF size of job is Very Small THEN change of lot is Large Positive
  4. IF size of job is Small THEN change of lot is Medium Positive
  5. IF size of job is Medium THEN change of lot is Medium Negative
  6. IF size of job is Large THEN change of lot is Large Negative
  7. IF urgency of job is High THEN change of lot is Large Positive
  8. IF urgency of job is Medium THEN change of lot is Medium Positive
  9. IF urgency of job is Low THEN change of lot is Large Negative

Figure 5. Fuzzy rules for deriving the size of the lots

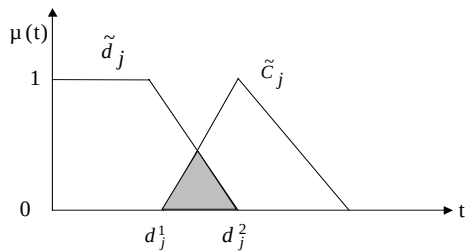


Figure 6. Satisfaction grade of completion time using area of intersection

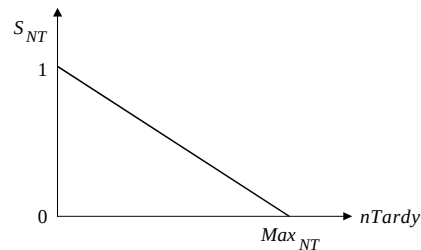


Figure 7. Satisfaction grade of Number of Tardy Jobs

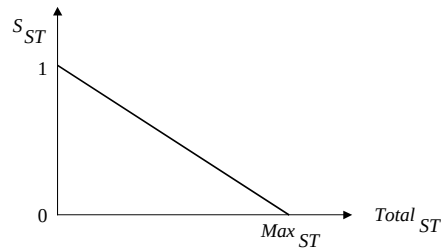


Figure 8. Satisfaction grade of Total Setup Time

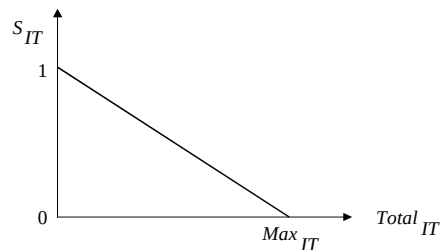


Figure 9. Satisfaction grade of Total Idle Time

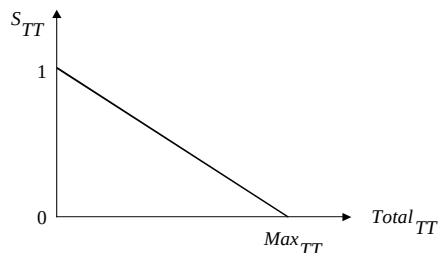


Figure 10. Satisfaction grade of Total Throughput Time

# A HISTORY OF DECISION-MAKING TOOLS FOR PRODUCTION SCHEDULING

Jeffrey W. Herrmann

*Department of Mechanical Engineering and Institute for Systems Research,  
University of Maryland, College Park, MD 20742, USA*

**Abstract:** Production scheduling is an important decision-making process that has embraced technology as computers and information systems became cheaper and easier to use. The history of production scheduling is not one of replacing human decision-makers with algorithms, however. This paper provides a historical perspective on the decision support tools that have been developed to improve production scheduling.

**Key words:** production scheduling, history, Gantt chart, scheduling systems

## 1. INTRODUCTION

Manufacturing facilities are complex, dynamic, stochastic systems. From the beginning of organized manufacturing, workers, supervisors, engineers, and managers have developed many clever and practical methods for controlling production activities.

Although dispatching rules, kanban cards, and other decentralized production control policies are in use, many manufacturing facilities generate and update production schedules, which are plans that state when certain controllable activities (e.g., processing of jobs by resources) should take place. Dispatching rules are usually quick but myopic because they typically they do not use global information. Production schedules can enable better coordination to increase productivity and minimize operating costs. A production schedule can identify resource conflicts, control the release of jobs to the job shop, ensure that required raw materials are ordered in time, determine whether delivery promises can be met, and identify time periods available for preventive maintenance.

The two key problems in production scheduling are, according to Wight (1984), “priorities” and “capacity.” In other words, “What should be done first?” and “Who should do it?” Wight defines *scheduling* as “establishing the timing for performing a task” and observes that, in manufacturing firms, there are multiple types of scheduling, including the detailed scheduling of a shop order that shows when each operation must start and complete. Cox *et al.* (1992) define *detailed scheduling* as “the actual assignment of starting and/or completion dates to operations or groups of operations to show when these must be done if the manufacturing order is to be completed on time.” They note that this is also known as *operations scheduling*, *order scheduling*, and *shop scheduling*. This paper is concerned with this type of scheduling.

Production scheduling is a complex decision-making process. Research scientists, software companies, and manufacturing consultants have developed and implemented advanced scheduling systems to reduce the effort of production scheduling and generate better schedules. These scheduling systems include computer algorithms that exploit results from scheduling theory and advanced optimization techniques.

Production scheduling involves a system of information gathering, decision-making, and schedule dissemination. Moreover, because the manufacturing system is dynamic and unexpected events occur, rescheduling is necessary because the production schedule is a plan that must be updated when the state of the manufacturing system makes the current production schedule infeasible. There are many types of disturbances that can upset the plan, including machine failures, processing time delays, rush orders,

quality problems, and unavailable material. See Vieira *et al.* (2003) for a more in-depth discussion of rescheduling.

Formal production scheduling systems will typically define a rescheduling period based on management measurement periods (one week or one day or one shift). But rescheduling also occurs (perhaps informally) during this period as unexpected events occur. As described by Wight and others, an informal scheduling process becomes necessary because the formal system cannot reschedule quickly enough to keep up with all of the different unexpected events that occur. The expeditors (who check inventory, call suppliers, and stage material) work in the informal system. Task start times are delayed, jobs are reassigned to different resources, and other adjustments take place. Intuitively, one can see that while small disruptions may be ignored in an informal way, larger disruptions may require significant changes to the production schedule to maintain good system performance. Huge disruptions will require many changes.

This paper discusses the history of tools used to support decision-making in real-world production scheduling. This story goes from the first charts developed by Henry Gantt to advanced scheduling systems that rely on sophisticated software. The goal of the paper is to help production schedulers, engineers, and researchers understand the true nature of production scheduling in dynamic manufacturing systems and to encourage them to consider how production scheduling systems can be improved even more. This paper includes material from a wide variety of articles and books on production scheduling to demonstrate the timeless importance of production scheduling and the range of approaches taken to improve it.

This paper does not address the sequencing of parts processed in high-volume, repetitive manufacturing systems. In such settings, one can look to JIT and lean manufacturing principles for how to control production. These approaches generally do not need the same type of production schedules discussed here. Although project scheduling will be discussed, the paper is primarily concerned with the scheduling of manufacturing facilities, not general project management. Note finally that this paper is not a review of the scheduling literature, which would take an entire volume.

The remainder of this paper is organized as follows: Section 2 discusses production scheduling prior to the advent of scientific management. Section 3 describes the first formal methods for production scheduling, many of which are still used today. Section 4 describes the rise of computer algorithms used for scheduling. Section 5 concludes the paper.

## 2. EARLY PRODUCTION SCHEDULING

Although humans have been creating items for countless years, manufacturing facilities first appeared during the First Industrial Revolution, when centralized power sources made new organizational structures viable. Hounshell (1984) provides a detailed look at the development of manufacturing technology in the United States of America. Wilson (2000a) provides an overview of manufacturing management and notes how modern manufacturing organizations developed from the mills and workshops and projects of the past. Unfortunately, neither of these excellent sources discusses the scheduling function in detail. Hopp and Spearman (1996) also provide a general overview of manufacturing in America since the First Industrial Revolution. McKay (2003) provides a historical overview of the key concepts behind the practices that manufacturing firms have adopted in modern times, highlighting, for instance, how the ideas of just-in-time (though not the term) were well-known in the early twentieth century.

The first factories were quite simple, and production scheduling started simply also. Schedules, when used at all, listed only when work on an order should begin or when the order is due. They didn't provide any information about how long the total order should take or about the time required for individual operations (Roscoe and Freark, 1971). This type of schedule was widely used before useful formal

methods became available (and can still be found in some small or poorly run shops). Binsse (1887) described a method for keeping track of time using a form almost like a Gantt chart.

Informal methods, especially expediting, have not disappeared. Wight (1984) stated that “production and inventory management in many companies today is really just order launching and expediting.” This author’s observation is that the situation has not changed much in the last 20 years.

### 3. THE INTRODUCTION OF FORMAL METHODS

Frederick Taylor’s separation of planning from execution justified the use of formal scheduling methods, which became critical as manufacturing organizations grew in complexity. Wilson (2000b) gives an interesting overview of the production planning office proposed by Taylor around the time of World War I. Many individuals were required to create plans, manage inventory, and monitor operations. (Computers would take over many of these functions decades later.) The “production clerk” created a master production schedule based on firm orders and capacity. The “order of work clerk” issued shop orders and released material to the shop.

The man uniquely identified with production scheduling is, of course, Henry L. Gantt, who created innovative charts for production control. According to Cox *et al.* (1992): a *Gantt chart* is “the earliest and best known type of control chart especially designed to show graphically the relationship between planned performance and actual performance.” However, it is important to note that Gantt created many different types of charts. Moreover, Gantt designed his charts so that foremen or other supervisors could quickly know whether production was on schedule, ahead of schedule, or behind schedule. Modern project management software includes this critical function even now.

Gantt (1903) describes two types of “balances”: the “man’s record,” which shows what each worker should do and did do, and the “daily balance of work,” which shows the amount of work to be done and the amount that is done. Gantt gives an example with orders that will require many days to complete. The daily balance has rows for each day and columns for each part or each operation. At the top of each column is the amount needed. The amount entered in the appropriate cell is the number of parts done each day and the cumulative total for that part. Heavy horizontal lines indicate the starting date and the date that the order should be done. According to Gantt, the graphical daily balance is “a method of scheduling and recording work.” In this article, Gantt also describes the use of production cards for assigning work to each operator and recording how much was done each day.

In *Work, Wages, and Profits* (originally published in 1916), Gantt explicitly discusses scheduling, especially in the job shop environment. He proposes giving to the foreman each day an “order of work” that is an ordered list of jobs to be done that day. Moreover, he discusses the need to coordinate activities to avoid “interferences.” However, he also warns that the most elegant schedules created by planning offices are useless if they are ignored, a situation that he observed.

In *Organizing for Work* (originally published in 1919), Gantt gives two principles for his charts: one, measure activities by the amount of time needed to complete them; two, the space on the chart can be used to represent the amount of the activity that should have been done in that time. Gantt shows a progress chart that indicates for each month of the year, using a thin horizontal line, the number of items produced during that month. In addition, a thick horizontal line indicates the number of items produced during the year. Each row in the chart corresponds to an order for parts from a specific contractor, and each row indicates the starting month and ending month of the deliveries. It is the closest thing to the Gantt charts typically used today in scheduling systems, though it is at a higher level than machine scheduling.

Gantt’s machine record chart and man record chart are quite similar, though they show both the actual working time for each day and the cumulative working time for a week. Each row of the chart

corresponds to an individual machine or operator. These charts do not indicate which tasks were to be done, however.

Clark (1942) provides an excellent overview of the different types of Gantt charts, including the machine record chart and the man record chart, both of which record past performance. Of most interest to those studying production scheduling is the *layout chart*, which specifies “when jobs are to be begun, by whom, and how long they will take.” Thus, the layout chart is also used for scheduling (or planning). The key features of a layout chart are the set of horizontal lines, one line for each unique resource (e.g., a stenographer or a machine tool), and, going across the chart, vertical lines marking the beginning of each time period. A large “V” at the appropriate point above the chart marks the time when the chart was made. Along each resource’s horizontal line are thin lines that show the tasks that the resource is supposed to do, along with each task’s scheduled start time and end time. For each task, a thick line shows the amount of work done to date. A box with crossing diagonal lines shows work done on tasks past their scheduled end time. Clark claims that a paper chart, drawn by hand, is better than a board, as the paper chart “does not require any wall space, but can be used on a desk or table, kept in a drawer, and carried around easily.” However, it is important to note that a chart carried and viewed by only one person is not a useful tool for communication.

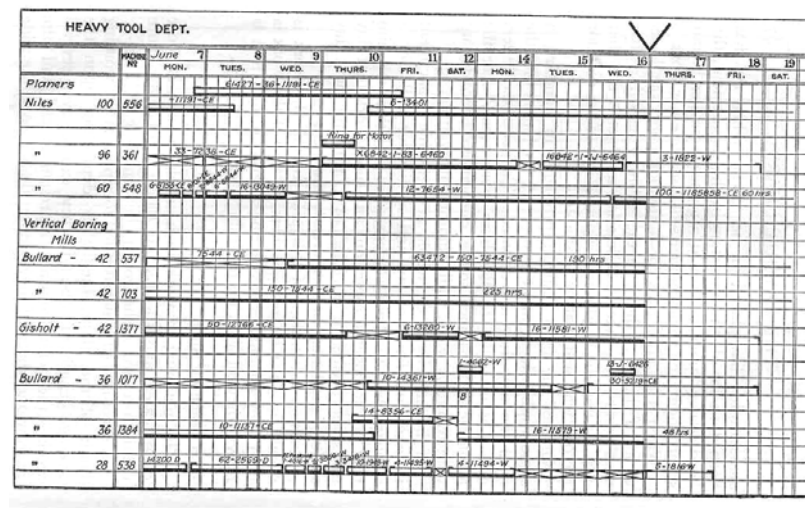


Figure 1. A Gantt Layout Chart (from Clark, 1942).

In conclusion, it can be said that Gantt was a pioneer in developing graphical ways to visualize schedules and shop status. He used time (not just quantity) as a way to measure tasks. He used horizontal bars to represent the number of parts produced (in progress charts) and to record working time (in machine records). His progress (or layout) charts had a feature found in project management software today: the length of the bars (relative to the total time allocated to the task) showed the progress of tasks.

Many firms implemented Taylor’s suggestion to create a production planning office, and production planners modified Gantt’s charts over the years. Mitchell (1939) discusses the role of the production planning department, including routing, dispatching (issuing shop orders) and scheduling. Scheduling is defined as “the timing of all operations with a view to insuring their completion when required.” Mitchell emphasizes that, in some shops, the shop foremen may be responsible for determining which specific worker and machine does which task. In others, the scheduling personnel have already determined this. The foreman, of course, has more insight into the qualitative factors that affect production. Mitchell describes two types of Gantt charts as typical of the graphical devices used to help those involved in scheduling. The Gantt load chart shows (as horizontal lines) the schedule of each machine and the total



load on the machine to date. Mitchell's illustration of this doesn't indicate which shop orders are to be produced. The Gantt progress chart shows (as horizontal lines) the progress of different shop orders and their due dates.

Muther (1944) describes scheduling in job shops, saying that foremen decide which work to do and then assign it to operators. Muther discusses the system used to schedule automobile manufacturing, including the dispatching of orders to purchasing, the body plant, the assembly lines, and the shipping department. However, since most production is done on assembly lines, detailed production schedules are not used. Muther also shows a schedule chart used to plan and track tasks for a specific job. Various horizontal bars show the start and end of subassembly tasks, and vertical bars show when subassemblies should be brought together.

MacNiece (1951) begins his discussion of scheduling with loading, which assigns an operation to a specific day or week when the machine (or machine group) will perform it. This loading is finite since it takes into account the number of machines, shifts per day, working hours per shift, days per week as well as the time needed to complete the order. MacNiece also gives a beautiful example of using a Gantt (layout) chart to solve a scheduling problem. The problem is to determine if an order for an assembly can be completed in 20 weeks. The Gantt chart has a row for each machine group and bars representing already planned work to which he adds the operations needed to complete the order. He argues that using a Gantt chart is a much quicker way to answer the question.

Roscoe and Freark (1971) also provide an example of a Gantt chart. It is a graphical schedule that lists the operations needed to complete an order. Each row corresponds to a different operation and lists the machine that will perform the operation and the rate at which the machine can produce parts (parts per hour), from which one can calculate the time required for that operation. Each column in the chart corresponds to a day, and each operation has a horizontal line from the day and time it should start to the day and time it should complete. The chart is used for measuring progress, so a thicker line parallel to the first line shows the progress on that operation to date. The authors state that a "Gantt chart is essentially a series of parallel horizontal graphs which show schedules (or quotas) and accomplishment plotted against time."

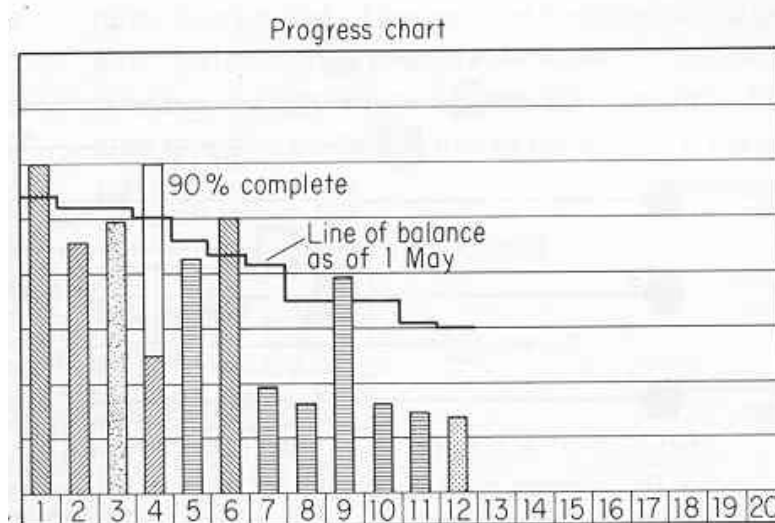


Figure 2. A Line of Balance (from O'Brien, 1969).

The *line of balance* can be used for determining how far ahead (or behind) a shop might be at producing a number of identical assemblies required over time. Given the demand for end items and a

bill-of-materials with lead times for making components and completing subassemblies, one can calculate the cumulative number of components, subassemblies, and end items that should be complete at a point in time to meet the demand. This line of balance is used on a progress chart that compares these numbers to the number of components, subassemblies, and end items actually done by that point in time (see Figure 2). The underlying logic is similar to that used by MRP systems, though this author is unaware of any scheduling system that use a line of balance chart today. More examples can be found in O'Brien (1969) and *Production Scheduling* (1973).

Control boards are another important type of scheduling tool. Cox *et al.* (1992) state that a *control board* is “a visual means of showing machine loading or project planning.” This is also called a *dispatching board*, a *planning board*, or a *schedule board*. MacNiece attributes planning boards to Taylor. The board described has one row of spaces for each machine. Each space represents one shift and contains one or more cards corresponding to the order(s) that should be produced in that shift, given capacity constraints. A large order requires multiple cards that are placed in consecutive spaces.

The Planalog control board was a sophisticated version developed in the 1960s. A Planalog is a board (up to six feet wide) that hangs on a wall (see Figure 3). The board has numerous rows into which one can insert gauges of different lengths (from 0.25 to 5 inches long). Each gauge represents a different task (while rows do not necessarily represent resources). The length of each gauge represents the task's expected (or actual) duration. The Planalog includes innovative “fences.” Each fence is a vertical barrier that spans multiple rows to show and enforce the precedence constraints between tasks. Moving a fence due to the delay of one task pushes all subsequent dependent tasks simultaneously.

#### 4. THE RISE OF COMPUTER ALGORITHMS

Unlike production scheduling in a busy factory, planning a large construction or systems development project is a problem that one can formulate and try to optimize. Thus, it is not surprising that large project scheduling was the first type of scheduling to use computer algorithms successfully.

O'Brien (1969) gives a good overview of the beginnings of the critical path method (CPM) and the Performance Evaluation and Review Technique (PERT). Formal development of CPM began in 1956 at Du Pont, whose research group used a Remington Rand UNIVAC to generate a project schedule automatically from data about project activities. In 1958, the development of PERT started in the office managing the development of the Polaris missile (the U.S. Navy's first submarine-launched ballistic missile). The program managers wanted to use computers to plan and monitor the Polaris program. By the end of 1958, the Naval Ordnance Research Calculator, the most powerful computer in existence at the time, was programmed to implement the PERT calculations. Both CPM and PERT are now common tools for project management.

Computer-based production scheduling emerged later. Wight (1984) lists three key factors that led to the successful use of computers in manufacturing:

1. IBM developed the Production Information and Control System starting in 1965.
2. The implementation of this and similar systems led to practical knowledge about using computers.
3. Researchers systematically compared these experiences and developed new ideas on production management.

O'Brien (1969) describes these early computer-based production scheduling systems, which automated the data collection and processing functions in existence since Taylor's day.

Wight (1984) also describes the success of material requirements planning (MRP), an approach that perfectly suited the computers in use at the time of its development. MRP, in turn, led to the rise of manufacturing resources planning (MRP II) and now enterprise resource planning (ERP) systems.

Rondeau and Litteral (2001) have described the history of MRP and ERP systems. For more about modern systems, see, for instance, Vollmann, Berry, and Whybark (1997).

Interactive, computer-based scheduling eventually emerged from various research projects to commercial systems. Godin (1978) describes many prototype systems. Duersch and Wheeler (1981) describe an early interactive computer-based scheduling program. The program, designed for assembly line production planning, could output graphs of monthly production and inventory levels on a computer terminal to help the scheduling personnel make their decisions. The software used standard strategies to generate candidate schedules that the scheduling personnel modified as needed. The software's key benefit was to reduce the time needed to develop a schedule. Adelsberger and Kanet (1991) use the term *leitstand* to describe an interactive production scheduling decision support system with a graphical display, a database, a schedule generation routine, a schedule editor, and a schedule evaluation routine. By that time, commercial leitstands were available, especially in Germany. The emphasis on both creating a schedule and monitoring its progress (planning and control) follows the principles of Henry Gantt. Similar types of systems are now part of modern manufacturing planning and control systems and enterprise resource planning systems.

Modern computer-based scheduling systems offer numerous features for creating, evaluating, and manipulating production schedules. Seyed (1995) provides a discussion on how to choose a system based on these features. Yen and Pinedo (1994) list the three primary components of a scheduling system: the database, the scheduling engine, and the user interface. The scheduling system may share a database with other manufacturing planning and control systems such as MRP or may have its own database, which may be automatically updated from other systems such as the manufacturing execution system. The user interface typically offers numerous ways to view schedules, including Gantt charts, dispatch lists, charts of resource utilization, and load profiles. The scheduling engine generates schedules and may use heuristics, a rule-based approach, optimization, or simulation. This framework is generally valid, but there exists a wide variety of scheduling software. Organizations such as APICS provide updated information about scheduling software and the vendors who sell them. McKay and Wiers (2004) provide practical guidelines on selecting and implementing scheduling software, and the challenge of implementing effective scheduling systems remains, as it did in Gantt's day (see, for instance, Yen and Pinedo, 1994, or Ortiz, 1996).

In some cases, manufacturing firms have created innovative scheduling systems to meet their particular needs. Each of these systems formulates the problem in a unique way that reflects each firm's specific scheduling objectives, and the system collects, processes, and generates information as part of a larger system of decision-making. Moreover, many years of research on optimization methods have created a large set of powerful algorithms that can be applied to generate schedules, from mathematical programming to searches that use concepts from artificial intelligence. Because a comprehensive review is not possible in this paper, a few examples will be mentioned.

Katok and Ott (2000) use mathematical programming to create a weekly schedule for an aluminum can manufacturing facility. Kuchta et al. (2004) use a mathematical programming approach to develop production schedules for mining at one of the world's largest underground mines. Numao (1994) describes production scheduling in a steel-making plant. Dawande et al. (2004) use a heuristic based on matching and bin packing to solve a slab design problem for a large steel plant.

Not all firms have embraced computer-based scheduling systems. Based on their survey of hundreds of manufacturing facilities, LaForge and Craighead (1998) conclude that computer-based scheduling can be successful if it uses finite scheduling techniques and if it is integrated with the other manufacturing planning systems. Computer-based scheduling can help manufacturers improve on-time delivery, respond quickly to customer orders, and create realistic schedules. Finite scheduling means using actual shop floor conditions, including capacity constraints and the requirements of orders that have already been released. However, only 25% of the firms responding to their survey used finite scheduling for part or all of their operations. Only 48% of the firms said that the computer-based scheduling system received

routine automatically from other systems. 30% said that a “good deal” of the data are entered manually, and 21% said that all data are entered manually. Interestingly, 43% of the firms said that they regenerated their schedules once each day, 14% said 2 or 3 times each week, and 34% said once each week.

## 5. SUMMARY AND CONCLUSIONS

The large changes in production scheduling are due to two key events. The first is Henry Gantt’s creation of useful ways to understand the complex relationships between men, machines, orders, and time. The second is the overwhelming power of information technology to collect, visualize, process, and share data quickly and easily, which has enhanced all types of decision-making processes.

The bad news is that many manufacturing firms have not taken advantage of these developments. They produce goods and ship them to their customers, but the production scheduling system is a broken collection of independent plans that are frequently ignored, periodic meetings where unreliable information is shared, expeditors who run from one crisis to another, and ad-hoc decisions made by persons who cannot see the entire system. Production scheduling systems rely on human decision-makers, and many of them need help.

This overview of production scheduling tools should be useful to those just beginning their study of manufacturing planning and control. This paper is the only known overview, and it collects descriptions of production scheduling from over 100 years. The author hopes that practitioners and researchers will use this overview to consider what has been truly useful to improve production scheduling practice in the real-world.

## ACKNOWLEDGEMENTS

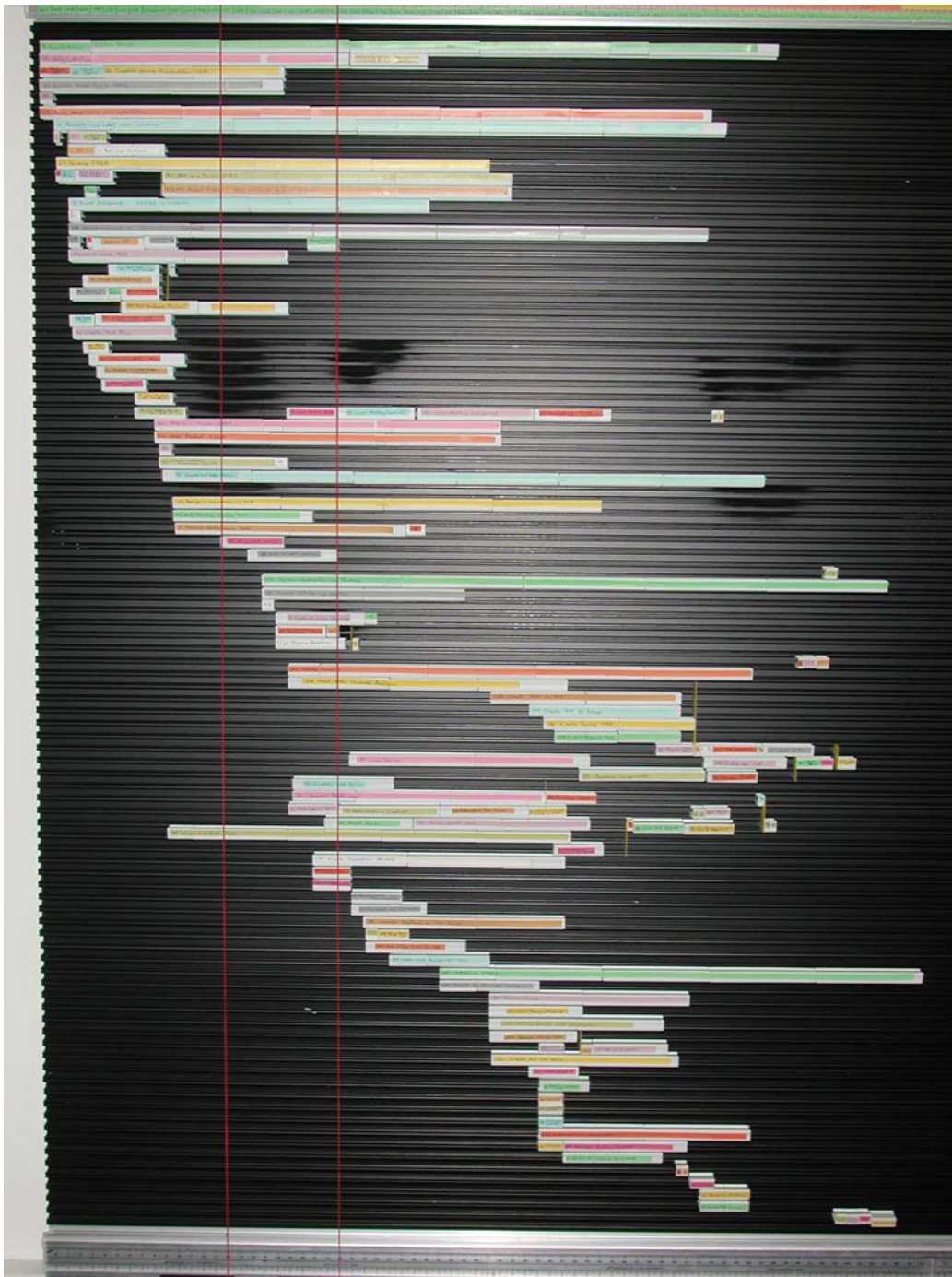
The author wishes to thank the students and colleagues, especially Robert Youker and John Kanet, who have contributed many valuable insights about the history of production scheduling. The author also thanks the anonymous reviewers for their very helpful comments and suggestions.

## REFERENCES

- Adelsberger, H., J. Kanet, The Leitstand - A New Tool for Computer Integrated Manufacturing, *Production and Inventory Management Journal*, Volume 32, Number 1, pages 43-48, 1991.
- Clark, Wallace, *The Gantt Chart, a Working Tool of Management*, second edition, Sir Isaac Pitman & Sons, Ltd., London, 1942.
- Cox, James F., John H. Blackstone, Jr., and Michael S. Spencer, editors, *APICS Dictionary*, American Production and Inventory Control Society, Falls Church, Virginia, 1992.
- Binsse, Henry L., A short way to keep time and cost, *Transactions of the American Society of Mechanical Engineers*, Volume 9, pages 380-386, 1887.
- Dawande, M., Kalagnanam, J., Lee, H. S., Reddy, C., Siegel, S., and Trumbo, M., 2004, The slab-design problem in the steel industry, *Interfaces* **34**(3):215-225.
- Duersch, Ralph R., and Dean B. Wheeler, An interactive scheduling model for assembly-line manufacturing, *International Journal of Modelling & Simulation*, Volume 1, Number 3, pages 241-245, 1981.
- Gantt, Henry L., A graphical daily balance in manufacture, *Transactions of the American Society of Mechanical Engineers*, Volume XXIV, pages 1322-1336, 1903.
- Gantt, Henry L., *Organizing for Work*, Harcourt, Brace, and Howe, New York, 1919. Reprinted by Hive Publishing Company, Easton, Maryland, 1973.
- Gantt, Henry L., *Work, Wages, and Profits*, second edition, Engineering Magazine Co., New York, 1916. Reprinted by Hive Publishing Company, Easton, Maryland, 1973.
- Godin, Victor B., Interactive scheduling: historical survey and state of the art, *AIIE Transactions*, Volume 10, Number 3, pages 331-337, 1978.

- Hopp, Wallace J., and Mark L. Spearman, *Factory Physics*, Irwin/McGraw-Hill, Boston, 1996.
- Hounshell, David A., *From the American System to Mass Production 1800-1932*, The Johns Hopkins University Press, Baltimore, 1984.
- Katok, E., and Ott, D., 2000, Using mixed-integer programming to reduce label changes in the Coors aluminum can plant, *Interfaces*, **30**(2):1-12.
- Kuchta, M., Newman, A., and E. Topal, 2004, Implementing a production schedule at LKAB's Kiruna mine, *Interfaces*, **34**(2):124-134.
- LaForge, R. Lawrence, and Christopher W. Craighead, *Manufacturing scheduling and supply chain integration: a survey of current practice*, American Production and Inventory Control Society, Falls Church, Virginia, 1998.
- MacNiece, E.H., *Production Forecasting, Planning, and Control*, John Wiley & Sons, Inc., New York, 1951.
- McKay, K.N., 2003, Historical survey of manufacturing control practices from a production research perspective, *International Journal of Production Research*, **41**(3):411-426.
- McKay, K.N., and Wiers, V.C.S., 2004, *Practical Production Control: a Survival Guide for Planners and Schedulers*, J. Ross Publishing, Boca Raton, Florida. Co-published with APICS.
- Muther, Richard, *Production-Line Technique*, McGraw-Hill Book Company, New York, 1944.
- Numao, M., 1994, Development of a cooperative scheduling system for the steel-making process, in *Intelligent Scheduling*, Zweben, M., and Fox, M.S., eds., Morgan Kaufmann Publishers, San Francisco.
- O'Brien, James J., *Scheduling Handbook*, McGraw-Hill Book Company, New York, 1969.
- Ortiz, Celia, Implementation issues: a recipe for scheduling success, *IIE Solutions*, pages 29-32, March, 1996.
- Production Scheduling*, American Institute of Certified Public Accountants, New York, 1973.
- Rondeau, Patrick J., and Lewis A. Litteral, Evolution of manufacturing planning and control systems: from reorder point to enterprise resource planning, *Production and Inventory Management Journal*, Second Quarter, pages 1-7, 2001.
- Roscoe, Edwin Scott, and Dorman G. Freark, *Organization for Production*, fifth edition, Richard D. Irwin, Inc., Homewood, Illinois, 1971.
- Seyed, Javad, Right on schedule, *OR/MS Today*, pages 42-44, December, 1995.
- Vieira, G.E., J.W. Herrmann, and E. Lin, Rescheduling manufacturing systems: a framework of strategies, policies, and methods, *Journal of Scheduling*, Volume 6, Number 1, pages 35-58, 2003.
- Vollmann, Thomas E., William L. Berry, and D. Clay Whybark, *Manufacturing Planning and Control Systems*, fourth edition, Irwin/McGraw-Hill, New York, 1997.
- Wight, Oliver W., *Production and Inventory Management in the Computer Age*, Van Nostrand Reinhold Company, Inc., New York, 1984.
- Wilson, James M., History of manufacturing management, in *Encyclopedia of Production and Manufacturing Management*, Paul M. Swamidass, ed., Kluwer Academic Publishers, Boston, 2000a.
- Wilson, James M., Scientific management, in *Encyclopedia of Production and Manufacturing Management*, Paul M. Swamidass, ed., Kluwer Academic Publishers, Boston, 2000b.
- Yen, Benjamin P.-C., and Michael Pinedo, On the design and development of scheduling systems, *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 197 – 204, October 10-12, 1994.

Figure 3. Planalog control board.



# **Real World Scheduling**

Abstracts

## Challenges in Matching Assets with Demand across All Supply Chain Management Decision Tiers in the IBM Microelectronics Division from Automated Fabs to Central Planning Engines to Collaborative Behavior (Work Arounds)

Kenneth Fordyce (contact author), Rich Burda, Alfred Degbotse, John Milne, James R. Morrison, and Elizabeth Williamson, IBM Systems and Technology Group

Every organization is faced with the challenge of responding to events in a synchronized and timely fashion. For example, a soccer team must constantly adjust as the ball and players change position. Each player must change in co-ordination with the other players and the team game plan. In the semiconductor industry, this response fundamentally involves matching an asset with a demand. Historically, this matching process was partitioned into one of four decision tiers (Kempf 1984, Sullivan 1984) called strategic (what markets will I be in and what do I need to do to be there); operational (how do I meet near term demand 1 week to 18 months out); tactical (what are the detail guidelines for execution patterns); and execution (actual assignment of lots to tools, shipping a lot, committing to an order). Additionally, the matching process between tiers was loosely coupled and involved a combination of decision support and human expertise.

Over the last few years, the IBM Microelectronics Division has recognized to obtain best of breed market responsiveness it must harness decision and information technology to tightly couple the tiers and automate the matching decisions. Three important subsets of this challenge are: (1) Automation (no human involved in the decision) of intelligently assigning lots to tools in a wafer fabricator where the product mix and business goals change rapidly, new equipment is modified and/or added regularly, and development and production lots cohabitate the fabricator. (2) Extending supply chain central planning engines to handle all complexities and intricacies of the enterprise such as request date versus commit date, pre-emptive demand priorities, extensive date effectively, and splintered product demand. (3) Automation of the dynamic adjusts to the division wide optimized central plan (currently handled by human collaborative behavior) to improve the plan (example: if an key order can not be met on time, capacity and/or lead time rules may be temporarily modified).

This presentation will provided a detailed description of the wafer fabricator dispatch and scheduling challenges at IBM's fully automated 300mm fab in East Fishkill, NY and partially automated 200mm fab in Burlington; evolution of thinking and experiences in dispatch scheduling covering such topics as rules versus optimization, flow control methods, changing business priorities, yields, advocates, zones of control; anticipated future direction; and key research challenges. Additionally, the challenges faced in (2) and (3) will be broadly described. Finally, the long term challenge of more tightly coupling the actions across the decision tiers will be described – the optimal matching decision at any decision tier is the one that best positions the enterprise to meet is prioritized demand – a little different then minimizing cycle time or maximizing throughput.



# **SCHEDULING WAFER FABRICATION OPERATIONS: STATE-OF-THE-PRACTICE AND FUTURE NEEDS**

John Fowler and Michele Pfund  
*Arizona State University, Tempe, USA*

**Abstract:** In this presentation, we provide an overview of the challenges in scheduling wafer fabrication operations, present the current state-of-the-practice, and discuss the needs that are not currently being met by today's practice. The material in this presentation is based on the results of a survey conducted of major semiconductor manufacturing companies.

**Key words:** Semiconductor Manufacturing, Wafer Fabrication, Scheduling, Dispatching.

## **1. OVERVIEW**

In order to understand the tools that are currently being utilized in semiconductor wafer fabrication facilities, a survey instrument was created and sent to major semiconductor manufacturing companies. The survey was designed to ask specific questions regarding the types of scheduling methodologies currently implemented, the limitations of these methodologies, and the needs for future generation scheduling systems. In total, 16 respondents from 14 companies participated in this survey, representing fabs from Europe, Asia, and North America.

From this report, we have found that many dispatching systems are in place and are mature installations (install time greater than 5 years). These systems have been considered to be only "satisfactory" in that benefits are being received, but the majority of respondents believe that more benefits are possible. Specifically, respondents indicated that better scheduling / dispatching rules, test environments, and reporting tools are needed.

Compared to the 1994 SEMATECH Measurement and Improvement of Manufacturing Capacity (MIMAC) survey and interview results, cycle time and on-time delivery have gained significant importance in the fab. Respondents indicated that these performance metrics are most impacted by a bottleneck machine breakdown and jobs going on hold (for process reasons), which were also the two most frequently occurring events as indicated by respondents. Thus, scheduling/rescheduling methodologies that incorporate these events are needed.

With regard to the frequency of rescheduling, respondents had mixed opinions. While most respondents favored rescheduling either at every shift or within a shift, many cited management challenges (operator stability/morale, staging for setup) as well as technical challenges such as hardware support. However, over 35% of respondents would like to reschedule after every job movement.

# **SCHEDULING PRODUCT FAMILIES IN A HIGH VOLUME, FLEXIBLE, ASSEMBLE-TO-ORDER FACTORY**

Susan K. Monkman<sup>1</sup>, Douglas J. Morrice<sup>2</sup>, and Jonathan F. Bard<sup>3</sup>

<sup>1,2</sup> *McCombs School of Business, The University of Texas at Austin*

<sup>3</sup> *Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin*

Key words: Production Scheduling, Real World Scheduling, Complexity of Scheduling Problems

## **1. ABSTRACT**

This research addresses and resolves a complex production scheduling problem that has arisen in a high-volume, assemble-to-order electronics manufacturer. Although this problem is found in the electronics industry, it arises out of conditions common to many industries. During the 1980s a major trend began in American manufacturing towards lean production facilities and flexible production lines. In recent years, there has also been a great proliferation of products targeting specific customer desires and an increase in the instances where customers can customize the product they order, leading to mass customization. These two trends combined result in increasing strain on manufacturing facilities to be even more flexible while producing high volumes of products within tight space constraints.

The problem studied here can be summarized as a problem of scheduling jobs from different product families on parallel assembly lines in order to minimize sequence-dependant setup costs but several special circumstances make this problem particularly complex and interesting. The factory assembles high volumes of products from many product families. A product family is defined by the chassis (frame) required by a product and the set of parts that might be used with that chassis type. Each family has some unique parts and some parts in common with other families. Each production line has space to hold three different types of chassis and many different types of parts. Therefore, each production line can produce products from any subset of three families, in any sequence, without requiring a setup. When a line must produce a product from a family whose parts are not on the line a setup must occur - removing some parts from the line to make room for the required parts. As a result, setup costs are sequence-dependent based on what set of families is currently on the line, which family's parts can be removed from the line, and which family's parts need to be placed on the line. There is not enough space on all the lines together to hold all the families' parts so setups are required.

In our research we chose to schedule the times that product families were on each production line, rather than schedule each job, for several reasons. First, this factory produces thousands of products each

day using a FIFO policy instead of order due dates with the goal of shipping orders as soon as possible. Second, setup costs depend only on which family is on the line, not which job is on the line. And third, due to the way order information arrives to the factory, the set of jobs to be completed is not known with certainty when the setups need to be planned. The proposed scheduling heuristic will produce a schedule that will allow production to be in nearly FIFO order, reduce production down-time for setups, and accommodate reasonable differences between actual orders to be produced and the order information known when the setups were planned. Although some modeling simplifications were made, the proposed heuristic can be used to resolve company's actual scheduling problem.

As mentioned in Pinto & Grossmann [1], there are three major components to a scheduling process: one is to assign tasks to equipment, another is to sequence the tasks, and the last is to schedule the exact time that each task will occur within the sequence. Our scheduling heuristic uses a combination of mixed integer programming and heuristics to achieve a low cost schedule. For the assignment component, we use a mixed integer program to assign product families to production lines based on expected demand in order to minimize the number of setups as well as achieve high part commonality between each set of families on the same line. Although this assignment model is NP hard, for the real problem size it can be solved in a reasonable amount of time.

For the sequencing component, we use another mixed integer program to sequence the families on each line. A typical sequencing problem with  $n$  product types on a single line can be modeled as a traveling salesman problem with  $n$  nodes where the distances between the nodes represent the setup costs. In our problem, the setup costs depend upon subsets of three families instead of one family, so we show how to model this sequencing problem as a traveling salesman subtour problem (TSSP) with  $nC_i$  nodes (i.e.  $n$  choose  $i$  nodes) where  $i$  is the number of families whose parts can fit on a single line. Here each node represents a possible subset of three product families. When a node is included on the selected path it indicates that the parts for that subset of three families will be on the assembly line together. In this representation there are  $n$  different overlapping subsets of nodes - one subset of nodes for each product family that consists of all the nodes that include that family. In a TSSP, every node does not get visited; rather, the goal is to find the shortest connected subtour that visits at least one node in each subset. In our case, this insures that every product family will be placed on a line at some point in the schedule so that products for that family can be assembled. Renaud and Boctor [2] provide a nice overview of particular cases of the TSSP, including the traveling purchaser problem and the covering salesman problem. Our problem is more complex than most, though, due to the fact that the subsets overlap very heavily: each node belongs to three different subsets. We develop a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic in order to find a good solution to the sequencing TSSP in a reasonable amount of time.

For the time-scheduling component, we use a heuristic to evenly distribute the time each family's parts spend on the line based on their expected demands and thus complete the schedule for when the parts for each subset of product families will be on each line. Products will then be dynamically scheduled for production in FIFO order during the time frame that the parts for their family type are on an assembly line.

*The authors would like to thank an anonymous reviewer for several helpful suggestions.*

## REFERENCES

- [1] J. M. Pinto and I. E. Grossmann, "Assignment and sequencing models for the scheduling of process systems," *Annals of Operations Research*, vol. 81, pp. 433-466, 1998.
- [2] J. Renaud and F. F. Boctor, "An efficient composite heuristic for the symmetric generalized traveling salesman problem," *European Journal of Operations Research*, vol. 108, pp. 571-584, 1998.

# Rostering

Full Papers

# A DECOMPOSITION, CONSTRUCTION AND POST-PROCESSING APPROACH FOR NURSE ROSTERING

Peter Brucker\*, Rong Qu†, Edmund Burke‡, Gerhard Post‡

\*University of Osnabrück, Faculty of Mathematic/Informatics, Albrechtstr. 28, 49069 Osnabrück, Germany, [peter@mathematik.uni-osnabrueck.de](mailto:peter@mathematik.uni-osnabrueck.de) †School of Computer Science & Information Technology, University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, UK, [{ekb, rxq}@cs.nott.ac.uk](mailto:{ekb, rxq}@cs.nott.ac.uk) ‡ ORTEC bv, Groningenweg 6-33, 2803 PV Gouda, The Netherlands

**Abstract:** This paper presents our work on decomposing a specific nurse rostering problem by cyclically assigning blocks of shifts, which are designed considering both hard and soft constraints, to groups of nurses. The rest of the shifts are then assigned to the nurses to construct a schedule based on the one cyclically generated by blocks. The schedules obtained by decomposition and construction can be further improved by a variable neighborhood search. Significant results are obtained and compared with a genetic algorithm and a variable neighborhood search approach on a problem that was presented to us by our collaborator, ORTEC bv, The Netherlands. We believe that the approach has the potential to be further extended to solve a wider range of nurse rostering problems.

**Key words:** nurse rostering problems, decomposition, variable neighborhood search

## 1. INTRODUCTION

Nurse rostering is concerned with highly constrained resource allocation problems. In hospitals and other medical institutes, the workload needs to be assigned to nurses periodically taking into account a number of constraints and requirements. *Hard* constraints are those that must be satisfied in order to have a *feasible* schedule due to physical resource restrictions and legislation. When requirements are desirable but not obligatory they are often referred to as *soft* constraints. Soft constraints are often used to evaluate the quality of feasible schedules.

Nurse rostering problems have a large number of variations on legal regulations and individual preferences, depending on countries and institutions. Typical issues concerned with nurse rostering include coverage demand, day-off requirements, weekend-off requirements, minimum and maximum workforce, etc.

A wide variety of different methodologies and models have been developed to deal with different problem circumstances during the years (see [13, 15, 19, 29]). These include mathematical programming (e.g. [4, 6, 26, 30, 31]), meta-heuristic methods (e.g. [2, 3, 9, 12, 18]) and constraint satisfaction techniques (e.g. [1, 8, 24, 25]). Meta-heuristic approaches which are particular appropriate for this paper because they employ variable neighborhood search for nurse rostering are [10, 11]. Column generation was employed in nurse scheduling (e.g. [5, 22]) using set covering-type models and integer programming. Knowledge based techniques have also been explored for solving nurse rostering problems (e.g. [7, 16, 23, 28]).

Over the years decomposition techniques have been applied successfully over a range of timetabling problems (e.g. [14, 21]). In [2] the authors developed a successful genetic algorithm where indirect representation was used. Different heuristic decoders were employed to construct the schedule, taking care of coverage and nurses' preferences in different stages. In [21] the constraints in the nurse scheduling problems in Japan were grouped into shift constraints and nurse constraints, based on which the problems were decomposed into sub-problems and which were solved repeatedly. In some of the staff scheduling problems, only predefined sequences of shifts, often referred as "stints" [19], are allowed to be used. Cyclical patterns were also studied in nurse rostering problems where each nurse receives cyclic scheduling each week for  $n$  weeks [13, 19, 20]. Pre and post processing methodologies for nurse rostering were presented and discussed in [17].

Our approach cyclically constructs partial solutions for groups of nurses based on pre-defined sequences of shifts in a specific nurse rostering problem taken from our commercial collaborator, ORTEC bv. The rest of the shifts are then assigned and the schedule is further improved by a post-processing procedure. It combines both the cyclic and non-cyclic schedules and aims at building near optimal sub-solutions while still being able to take care of individual requirements.

The problem being tested in this work has been addressed by the commercialized software HARMONY<sup>TM1</sup> [27], which has been developed by ORTEC, The Netherlands and is in use in hospitals, and other organizations in different countries. In this paper we propose a decomposition, construction and post-processing approach to solve this specific problem. The interface of HARMONY<sup>TM</sup> is used as a tool to visualize the schedules, to indicate infeasibilities, and to calculate penalties for violations of soft constraints.

## 2. THE NURSE ROSTERING PROBLEM

The nurse rostering problem we consider here is based on the situation encountered by ORTEC in intensive care units in Dutch hospitals. We outline the following characteristics.

1. We have to adhere to Dutch national legislation, and the collective labor agreements in force in Dutch hospitals. They are translated to a shorter period, to be meaningful for a situation without (much) history.
2. The requests of the personnel are very important, and should be met as much as possible; the soft constraints we use are those that, in our experience, represent the situation in Dutch hospitals.
3. It is not necessary to consider qualifications, as all personnel are highly qualified. Nurses still in training are disregarded in the original planning (and are added by hand afterwards).
4. The size of the department is rather small, but convenient for testing purposes.

The problem consists of assigning a certain number of different types of shifts (which are described in Section 2.1) within a scheduling period of 4 weeks to 16 nurses in a ward. Twelve of the nurses are full-time and have a contract of 36 hours per week. One part-time nurse works 32 hours per week and the other three part-time nurses work maximally 20 hours per week. The hard and soft constraints that need to be satisfied are described below in Sections 2.2 and 2.3, respectively. This problem is also presented in [10].

### 2.1 Shift Types and Demand

There are 4 different shift types in the problem. Table 1 gives the definitions and the daily coverage requirements for these shift types. Each of the shift types cover 9 hours including one hour of resting time, except the night shift, which contains no rest. So there are 8 actual working hours for each of these shift types.

### 2.2 Hard Constraints

The hard constraints listed below must be met in any circumstances otherwise the schedule is considered to be infeasible and unacceptable.

1. Demand needs to be fulfilled<sup>2</sup> (i.e. all the requested shifts in Table 1 must be covered).
2. For each day, 1 nurse may start only one shift.
3. Within a scheduling period, a nurse is allowed to exceed the number of hours for which he/she is available for his/her department by at most 4 hours.
4. The maximum labor time per week is on average 36 hours over a period of 13 consecutive weeks if this period does not include work during night shifts.
5. The maximum number of night shifts is 3 per period of 5 consecutive weeks.
6. A nurse must receive at least 2 weekends off duty per 5 week period. A weekend off duty lasts 60 hours including Saturday 00:00 to Monday 04:00.
7. Following a series of at least 2 consecutive night shifts, a 42 hours rest is required.

<sup>1</sup> [http://www.ortec.com/us/area.php?id=ORTEC\\_harmony](http://www.ortec.com/us/area.php?id=ORTEC_harmony)

<sup>2</sup> It would be possible to consider this as the main objective in the problem rather than as a hard constraint. However, we treat it as one of the hard constraints, as do many other papers in the nurse scheduling.

8. During any period of 24 consecutive hours, at least 11 hours of rest is required. A night shift has to be followed by at least 14 hours rest. An exception is that once in a period of 21 days for 24 consecutive hours, the resting time may be reduced to 8 hours.
- 9 The number of consecutive night shifts is at most 3.
- 10 The number of consecutive shifts (workdays) is at most 6.
- 11 One of the full-time nurses requires not receiving any late shifts.

## 2.3 Soft Constraints

Requirements, or soft constraints, and their weights in the problem we are dealing with are listed in Table 2. Ideally these constraints should be satisfied as much as possible. However, in real world circumstances, it is usually necessary to violate some of these soft constraints. Depending on how strongly these soft constraints are desired (especially in comparison to other soft constraints), a weight is assigned to each of them. The higher the weight, the more strongly desired the constraint is. For example, soft constraints 9 and 10 in Table 2 have very low weights as they are not so important and may be violated at low cost. The penalty of a feasible schedule is the sum of the weights of all the violations of soft constraints in the schedule.

## 3. A SOLUTION APPROACH

Our decomposition approach has two stages of solution construction and one stage of solution improvement. The first two stages are currently manually carried out and the 3<sup>rd</sup> stage is an automatic process implemented in [10]. They are described as below:

1. Firstly, sequences of shifts (we call them *blocks*) are designed taking into account both the soft and hard constraints. These blocks are then used to construct cyclic schedules for groups of nurses (we call the schedule for a group a *cyclic pattern*).
2. After the assignment by cyclic patterns, the rest of the shifts are assigned to nurses who are not involved in the cyclic patterns. In connection with this, some shifts assigned in the cyclic patterns may need to be re-assigned to accommodate the remaining shifts.
3. A variable neighborhood search approach developed in [10] is then applied to further improve the schedule obtained from the first two stages. It reassigns the shifts to improve the quality of the schedule.

### 3.1 Cyclic Pattern Design

In the first stage, cyclic patterns are constructed. Each cyclic pattern defines a schedule for 5 nurses over 5 weeks. Here a duration of 5 weeks is considered to be the length of the scheduling period due to hard constraints 5 and 6 (presented in Section 2.2). This duration can be adjusted to any other number of  $x$  weeks and the number of nurses can also be adjusted. The building elements of a cyclic pattern are called blocks (see below).

#### 3.1.1 Construction of Blocks

When designing blocks, we first consider shifts which are the most important or most difficult to be scheduled. In this problem, night shifts are the most important ones with a cost of 1000 if the length is not within the range of [2, 3] (soft constraint 3). Also shifts at weekends are important as a non-complete weekend will cause a cost of 1000 (soft constraint 1).

We therefore consider the night shifts first in designing the blocks. This will result in three basic shift patterns (presented in the lefthand-side of the table in Table 3). They are the only three valid assignments for night shifts satisfying soft constraints 1 and 3. An example of invalid basic patterns is shown in the righthand-side of the table in Table 3, from which we can see that in *Pattern e* a night shift on Friday ends at 7:00 am (as defined in Table 1) on Saturday, making the weekend not complete. This leads to the fact that the other two patterns *Pattern d* and *f* cannot be considered in designing the blocks as they have to be used in conjunction with *Pattern e*.



It can be observed from soft constraints 3 and 5-9 in Table 2 that good schedules will consist of blocks within certain length ranges. When designing the blocks based on the basic patterns presented in Figure 1, both the hard and soft constraints listed in Sections 2.2 and 2.3 are considered with. The aim is to avoid violations of hard constraints, and to minimize the costs of violations of soft constraints with high penalties (soft constraints 1-3 in Table 2) associated with each block.

Sequencing penalties within blocks are formulated and presented in Table 4, and are checked when designing the blocks. In the table the values are the penalties of the violations (weights of soft constraints). Here “n/f” indicates a non-feasible schedule if the shift type on the corresponding row is followed by the shift type on the corresponding column. For example, a night shift followed by an early shift (“n/f” at the 2<sup>nd</sup> column and the 1<sup>st</sup> row in Table 4) indicates a non-feasible schedule (hard constraint 8), and a day shift followed by an early shift (soft constraint 9a) causes a cost of 5 (at the 2<sup>nd</sup> column and the 3<sup>rd</sup> row in Table 4) in the schedule.

### 3.1.2 Building of Cyclic Patterns

Based on the blocks built by the above process, a set of cyclic patterns can be designed concerning both the hard and soft constraints. For example, the total number of hours during 5 weeks needs to be within the maximum working hours allowed, and the resting time after blocks needs to be not shorter than required, etc. The cyclic patterns with the least penalties (below a certain threshold) are used to construct the schedules for groups of 5 nurses in the first stage of the approach.

Besides trying to maximize the number of shifts within blocks (to leave as few as possible shifts to be assigned later), the balance of the coverage of D, E and L shift types are also concerned. They should be equally distributed so that the rest of shifts can also be distributed evenly within the schedule in the 2<sup>nd</sup> stage of approach. The total numbers of D, E and L shifts on the corresponding weekdays within a week in the upper table in Table 5 are presented in the lower table in Table 5. The block covers one E shift type each day except Wednesday, Saturday and Sunday, one D shift type almost equally each day except Friday, and one L shift type each day except Wednesday and Thursday, shown in the lower table in Table 5. The maximum number of N shift types within 5 weeks needs to be 3 (see hard constraint 5), thus there are only 3 N shifts in this cyclic pattern, as also shown in the lower table in Table 5.

The upper table in Table 5 presents an example of a schedule constructed by a block based on the basic pattern, *Pattern c*, presented in Table 3. The schedule of a group of 5 nurses for the 1<sup>st</sup> week is shown. For the 2<sup>nd</sup> week, for the same group of nurses, a cyclic shift is applied to their first week schedule (i.e. the schedule of nurse  $i$  is replaced by the schedule of nurse  $i + 1$  for  $i = 1, 2, 3, 4$  and the schedule of nurse 5 is replaced by the schedule of nurse 1). These cyclic exchanges are repeated four times.

Figure 1 presents a HARMONY<sup>TM</sup> screenshot of a schedule with two cyclic patterns in a nurse-day view for the problem we described in Section 2. As the number of full-time nurses in this problem is 12, two cyclic patterns are designed for 10 nurses (as shown in the figure). There are only three part-time nurses who work maximally 20 hours a week, and one part-time nurse who work 32 hours a week, so no cyclic patterns are designed for part-time nurses in this problem.

We can see that for nurse “Banar” (the 2<sup>nd</sup> nurse in the schedule in Figure 3), in the last week’s schedule there is one L (on Friday 28<sup>th</sup> February) less than the other 4 nurses with the same cyclic pattern. The same situation also applies for nurse “Navel” (the last nurse in the schedule in Figure 3), whose schedule in the last week (27<sup>th</sup> and 28<sup>th</sup> Thursday and Friday) is two L less than the others with the same cyclic pattern. The reason is that in HARMONY<sup>TM</sup>, the average maximum working hours for each nurse is calculated within a calendar month instead of 5 weeks in the problem presented. This adds an extra restriction to the problem so three shifts have to be removed from the schedule.

## 3.2 Construction of a Schedule

The shifts not covered by the cyclic patterns are then assigned into the schedule taking into account both the hard and soft constraints in the second stage of the approach. This may involve re-assigning some of the shifts already in the schedule.

Figure 2 presents a complete schedule by using the first two stages of (decomposition and construction) the overall approach proposed. The last column in the schedule gives the penalties for each of the nurses on the corresponding rows. The quality of the schedule is evaluated by summing up all the penalties on the last column, calculating the penalty of the overall schedule in Figure 2 for the problem to be 340.

At this stage we are only investigating the construction of the cyclic patterns and the construction of the schedules manually to have a good understanding of the potential and the limitations of the approach proposed.

### 3.3 Improving the Complete Schedule

The last stage of the approach employs a variable neighborhood search to further improve the schedules constructed by the first two stages (decomposition and construction). Please note that after the improvements made in the 3<sup>rd</sup> stage, the part of the schedule generated by the cyclic patterns may not be cyclic any more. The aim of the 1<sup>st</sup> stage of the approach is to build good sub-solutions when decomposing the problem and doing this with much less effort.

## 4. COMPARISONS AND DISCUSSION

Due to the large number of variants and different techniques investigated for nurse rostering problems, it is difficult to carry out fair and meaningful comparisons upon the computational effort taken and the solution quality obtained. However, it is possible to compare against the methods that are used to solve the same problem as that presented in this paper.

In the current version of HARMONY<sup>TM</sup>, a hybrid genetic algorithm is employed. A local search is carried out after each generation of the genetic algorithm to further improve the individual schedules to their local optima before the next generation. The moves taken in the local search include moving a shift assigned from one nurse to another, and swapping two single shifts assigned to two nurses.

A hybrid variable neighborhood search (VNS) has also been developed and applied to the same problem [10], where the same types of moves as in the above genetic algorithm are used within a descent local search on single schedules rather than a population of schedules at a time. When no further improvement can be made by using both moves, parts of the schedule which cause high penalties are re-assigned and the descent local search is started again. See [10] for more details.

Using the decomposition, construction and post-processing approach, we obtained a number of different schedules on the problem presented in Section 2. The best results out of 5 runs on each of the approaches, namely the hybrid genetic algorithm, the variable neighborhood search and our approach with and without the variable neighborhood search approach as the 3<sup>rd</sup> stage of post-processing, are presented in Table 6. The values in parentheses give the computational time of the corresponding approaches.

From the table we can observe that for the problem presented in this paper, the schedules generated by the hybrid GA have lower penalty than that of VNS but requiring much more computational time (6 hours as opposed to 1 minute). The decomposition, construction and post-processing approach obtained better results both with and without VNS. It is difficult to compare the computational time among the approaches as currently the design of blocks and cyclic patterns are made manually. The post-processing by variable neighborhood search usually stops within 1 minute.

It can be observed here that the decomposition and construction approach can serve as a good initialization method for searching algorithms such as VNS. The reason may be that pre-designed blocks and cyclic patterns can provide a good backbone in the schedule, based on which searching algorithms can start from promising areas in the search space, which is usually very huge and complex for nurse rostering problems. This observation leads us to hypothesize that this approach may produce good results on other problems and with other search algorithms.

## 5. CONCLUDING REMARKS

A nurse rostering problem presented by ORTEC has been solved by our decomposition, construction and post-processing approach. Blocks of shifts are designed to cyclically assign sequences of shifts, rather than single shifts, to groups of nurses. This will generate good partial solutions within the schedule, based on which the rest of the shifts are assigned afterwards. Both stages consider the hard and soft constraints and aim at minimizing the penalty of the schedule constructed. Extra constraints such as personal requests can be taken care of by the 2<sup>nd</sup> and 3<sup>rd</sup> stages of the approach, which are flexible enough to take care of other possible constraints or personal requests.

Our decomposition and construction approach is able to construct near optimal sub-solutions for sub-problems, which are obviously less complex than the overall problem. It combines the advantages of cyclical patterns and the non-cyclic schedules, which greatly reduce the computational time and take care of individual constraints/requests.

In our future work, the approach can be extended by employing heuristic approaches on automatically designing cyclic patterns, assigning the rest of the shifts, and improving the schedules generated. Based on the observations and experience collected in this work, the combinations between the patterns (not just in cyclic but rather at a general manner) will be further investigated. Column generation techniques will also be investigated in our future research.

Although our approach is tested on a specific problem, the idea of decomposition can be extended to solve a wider range of nurse rostering problems. Our future work will also include testing this approach on more problem instances and a wider range of problems to carry out more comparisons among different nurse rostering techniques.

## ACKNOWLEDGEMENTS

The work is supported by the UK Engineering & Physical Sciences Research Council (EPSRC) (grant number GR/T23374/01). Professor Brucker's contribution is supported by a EPSRC visiting fellowship (grant number GR/T23374/01). We would like to thank the anonymous referees' for their valuable comments. We would also like to acknowledge the valuable support from ORTEC bv, The Netherlands.

## REFERENCES

1. Adbennadher S. and Schlenker H.: Nurse Scheduling using Constraint Logic Programming, Eleventh Annual Conference on Innovative Applications of Artificial Intelligence, IAAI-99 (1999), 838-843.
2. Aickelin U. and Dowsland K.: An Indirect Genetic Algorithm for a Nurse Scheduling Problem. JORS, (2003) 31(5): 761-778.
3. Aickelin U. and White P.: Building Better Nurse Scheduling Algorithms. Annals of Operations Research, (2004), 128:159-177.
4. Bailey J. and Field J.: Personnel Scheduling with Flexshift Models. Journal of Operations Management, (1985) 5(3): 327-338.
5. Bard J. and Purnomo H.W.: Preference Scheduling for Nurses using Column Generation, EJOR, (2005), 164: 510-534.
6. Beaumont N.: Scheduling Staff using Mixed Integer Programming. EJOR, (1997) 98: 473-484.
7. Beddoe G. and Petrovic S.: Determining Feature Weights Using a Genetic Algorithm in a Case-Based Reasoning Approach to Personnel Rostering. Accepted for publication in EJOR, (2005).
8. Berrada I., Ferland J.A. and Michelon P.: A Multi-objective Approach to Nurse Scheduling with both Hard and Soft Constraints. Socio-Economic Planning Science, (1996), 30(3): 183-193.
9. Burke E., Cowling P., De Causmaecker P. and Vanden Berghe G.: A Memetic Approach to the Nurse Rostering Problem. Applied Intelligence, (2001) 15: 119-214.

10. Burke E., Curtois T., Post G., Qu R. and Veltman B. A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem. Technical Report NOTTCS-TR-2005-3, School of CSiT, University of Nottingham 2005.
11. Burke E., De Causmaecker P., Petrovic S. and Vanden Berghe G.: Variable Neighborhood Search for Nurse Rostering Problems, Ch. 7 in *Metaheuristics: Computer Decision-Making (Combinatorial Optimization Book Series)*, (eds. Resende M.G.C. and de Sousa J.P.), Kluwer, (2003), 153-172.
12. Burke E., De Causmaecker P. and Vanden Berghe G.: A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. *Selected Papers from the 2nd Asia Pacific Conference on Simulated Evolution and Learning (1998)*, LNAI 1585, 187-194.
13. Burke E., De Causmaecker P., Vanden Berghe G. and Van Landeghem H.: The State of the Art of Nurse Rostering. *Journal of Scheduling*, (2004) 7(6): 441-499.
14. Burke E. and Newall J.: A Multi-Stage Evolutionary Algorithm for the Timetable Problem, *IEEE Transactions on Evolutionary Computation*, (1999) 3(1): 63-74.
15. Cheang B., Li H., Lim A. and Rodrigues B.: Nurse Rostering Problems - a Bibliographic Survey. *EJOR*, (2003) 151: 447-460.
16. Chen J. and Yeung T.: Hybrid Expert System Approach to Nurse Rostering. *Computers in Nursing*, (1993) 11: 183-190.
17. De Causmaecker P. and Vanden Berghe G.: Relaxation of Coverage Constraints in Hospital Personnel Rostering. In: *Proceedings of the 4th International Conference on the Practice and Automated Timetabling (2002)*, 187-206.
18. Dowsland K.: Nurse Scheduling with Tabu Search and Strategic Oscillation. *EJOR*, (1998) 106: 393-407.
19. Ernst A.T., Jiang H., Krishnamoorthy M. and Sier D.: Staff Scheduling and Rostering: A Review of Applications, Methods and Models. *EJOR*, (2004) 153: 3-27.
20. Howell J.P.: Cyclical Scheduling of Nursing Personnel. *Hospital JAHA*, (1998) 40: 77-85.
21. Ikegami A. and Niwa A: A Subproblem-centric Model and Approach to the Nurse Scheduling Problem, *Mathematical Programming*, (2003), 97(3): 517-541.
22. Jaumard B., Semet F. and Vovor T.: A Generalized Linear Programming Model for Nurse Scheduling, *EJOR*, (1998), 107: 1-8.
23. Lukman D., May J. Shuman L. and Wolfe H.: Knowledge-Based Schedule Formulation and Maintenance Under Uncertainty. *Journal of the Society of Health Systems*, (1991) 2: 42-64.
24. Meyer H auf'm Hofe H.: Solving Rostering Tasks as Constraint Optimisation. In: *Selected Papers from the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT'00)*, LNCS 2079, (2000) 280-297.
25. Nonobe K. and Ibaraki T.: A Tabu Search Approach to the Constraint Satisfaction Problem as a General Problem Solver. *EJOR*, (1998) 106: 599-623.
26. Pierskalla W. and Rath G.: Nurse Scheduling using Mathematical Programming. *OR*, (1976) 24(5): 857-870.
27. Post G. and Veltman B.: Harmonious Personnel Scheduling. In: *Proceedings of the 5th International Conference on the Practice and Automated Timetabling (PATAT'04)*, 557-559.
28. Scoot S. and Simpson R.: Case-Bases Incorporating Scheduling Constraint Dimensions – Experiences in Nurse Rostering. In: *Advances in Case-Based Reasoning, European Workshop on Case Based Reasoning 1998*, LNAI 1488, (1998) 392-401.
29. Sitompul D. and Randhawa S.: Nurse Rostering Models: A State-of-the-Art Review. *Journal of the Society of Health Systems*. (1990) 2: 62-72.
30. Valouxis C. and Housos E.: Hybrid Optimisation Techniques for the Workshift and Rest Assignment of Nursing Personnel. *Artificial Intelligence in Medicine*, (2000), 20: 155-175.
31. Warner M.: Scheduling Nursing Personnel According to Nurse Preference: a Mathematical Programming Approach. *OR*, (1976) 24: 842-856.

Table 1. Shift types and demand during a week

| Shift type | Start time | End time | Demand |     |     |     |     |     |     |
|------------|------------|----------|--------|-----|-----|-----|-----|-----|-----|
|            |            |          | Mon    | Tue | Wed | Thu | Fri | Sat | Sun |
| Early      | 07:00      | 16:00    | 3      | 3   | 3   | 3   | 3   | 2   | 2   |
| Day        | 08:00      | 17:00    | 3      | 3   | 3   | 3   | 3   | 2   | 2   |
| Late       | 14:00      | 23:00    | 3      | 3   | 3   | 3   | 3   | 2   | 2   |
| Night      | 23:00      | 07:00    | 1      | 1   | 1   | 1   | 1   | 1   | 1   |

Table 2. Soft constraints and their weights

| Soft Constraints |                                                                                                                                                                                                         | Weights |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| 1                | For the period of Friday 23:00 to Monday 0:00, a nurse should have either no shifts or at least 2 shifts (Complete Weekend).                                                                            | 1000    |
| 2                | Avoid sequence of shifts with length of 1 for all nurses.                                                                                                                                               | 1000    |
| 3a               | For nurses with availability of 30-36 hours per week, the length of a series of <i>night</i> shifts should be within the range [2, 3]. It could be part of, but not before, another sequence of shifts. | 1000    |
| 3b               | For nurses with availability of 0-30 hours per week, the length of a series of <i>night</i> shifts should be within the range [2, 3]. It could be part of, but not before, another sequence of shifts.  | 1000    |
| 4                | The rest after a series of <i>day</i> , <i>early</i> or <i>late</i> shifts is at least 2 days.                                                                                                          | 100     |
| 5a               | For nurses with availability of 30-36 hours per week, the number of shifts is within the range [4, 5] per week.                                                                                         | 10      |
| 5b               | For nurses with availability of 0-30 hours per week, the number of shifts is within the range [2, 3] per week.                                                                                          | 10      |
| 6a               | For nurses with availability of 30-36 hours per week, the length of a series of shifts should be within the range of [4, 6].                                                                            | 10      |
| 6b               | For nurses with availability of 0-30 hours per week, the length of a series of shifts should be within the range [2, 3].                                                                                | 10      |
| 7                | For all nurse, the length of a series of <i>early</i> shifts should be within the range [2, 3]. It could be within another series of shifts.                                                            | 10      |
| 8                | For all nurse the length of a series of <i>late</i> shifts should be within the range of [2, 3]. It could be within another series of shifts.                                                           | 10      |
| 9a               | An <i>early</i> shift after a <i>day</i> shift should be avoided.                                                                                                                                       | 5       |
| 9b               | An <i>early</i> shift after a <i>late</i> shift should be avoided.                                                                                                                                      | 5       |
| 9c               | A <i>day</i> shift after a <i>late</i> shift should be avoided.                                                                                                                                         | 5       |
| 10               | A <i>night</i> shift after an <i>early</i> shift should be avoided.                                                                                                                                     | 1       |

Table 3. Basic night shift patterns

| Valid     |               | Invalid   |               |
|-----------|---------------|-----------|---------------|
|           | M T W T F S S |           | M T W T F S S |
| Pattern a | N N - - - - - | Pattern d | N N N - - - - |
| Pattern b | - - N N - - - | Pattern e | - - - N N - - |
| Pattern c | - - - - N N N | Pattern f | - - - - - N N |

Table 4. Penalties of the violations within blocks

|                 |   | Succeeding shifts |           |           |     |
|-----------------|---|-------------------|-----------|-----------|-----|
|                 |   | N                 | E         | D         | L   |
| Preceding shift | N | ok                | n/f       | n/f       | n/f |
|                 | E | ok                | ok        | ok        | ok  |
|                 | D | ok                | 5         | ok        | ok  |
|                 | L | ok                | n/f or 5* | n/f or 5* | ok  |

\*: depends on the scheduling period where the shifts are (hard constraint 8, or soft constraints 9b and 9c)

Table 5. A cyclic pattern of 5 weeks (upper table) and the coverage of different shift types (lower table).

|        | Week 1 |   |   |   |   |   |   | Week 2 |   |   |   |   |   |   |          |
|--------|--------|---|---|---|---|---|---|--------|---|---|---|---|---|---|----------|
|        | M      | T | W | T | F | S | S | M      | T | W | T | F | S | S |          |
| Nurse1 | D      | D | L | L | L |   |   |        |   | D | D | N | N | N | ... etc. |
| Nurse2 |        |   | D | D | N | N | N |        |   |   | E | E | D | D |          |
| Nurse3 |        |   |   | E | E | D | D | L      | L |   |   |   | L | L |          |
| Nurse4 | L      | L |   |   |   | L | L | E      | E | L | L |   |   |   |          |
| Nurse5 | E      | E | L | L |   |   |   | D      | D | L | L | L |   |   |          |

Cost for each nurse = 5

Violation: L is followed by E (soft constraint 9b in Table 2)

Number of shifts covered in one week

|   | M | T | W | T | F | S | S |
|---|---|---|---|---|---|---|---|
| N | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| E | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| D | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| L | 1 | 1 | 2 | 2 | 1 | 1 | 1 |

Table 6. Comparisons of the approaches (Hybrid GA: genetic algorithm + local search; Hybrid VNS: iterated variable neighborhood search, and our approaches) on the problem.

| Hybrid GA        | Hybrid VNS     | Decomposition<br>+ construction | Decomposition +<br>construction + VNS |
|------------------|----------------|---------------------------------|---------------------------------------|
| 630 (5 minutes)  | 466 (1 minute) | 340                             | 170 (< 1 minute)                      |
| 505 (40 minutes) | -              | -                               | -                                     |
| 411 (6 hours)    | -              | -                               | -                                     |



Figure 1. Blocks assigned to 2 groups of 5 nurses, respectively

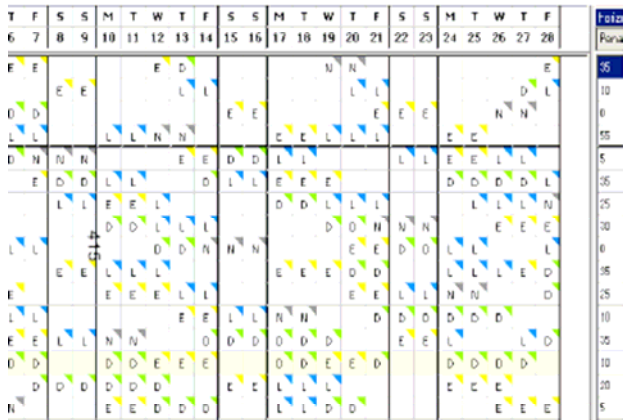


Figure 2. A complete schedule using the decomposition approach

# **Rostering**

Abstracts



# Integrating nurse and surgery scheduling

Jeroen Beliën and Erik Demeulemeester

*Department of Applied Economics*

*Katholieke Universiteit Leuven*

*Naamsestraat 69, 3000 Leuven, Belgium*

jeroen.beliën@econ.kuleuven.be

erik.demeulemeester@econ.kuleuven.be

## 1. Introduction and scope

One common problem at hospitals is the extreme variation in daily (even hourly) workload pressure for nurses. The operating room is considered to be the main engine and hence the main generator of variance in the hospital. It is our belief that integrating the operation room scheduling process with the nurse scheduling process is a simple, yet effective way to achieve considerable savings in staffing costs. The purpose of this paper is threefold. First of all, we present a concrete model that integrates both the nurse and the operating room scheduling process. Secondly, we show how the column generation technique approach, often employed for nurse scheduling problems, can easily cope with this model extension. Thirdly, by means of a large number of computational experiments we provide an idea of the cost saving opportunities and required solution times.

## 2. Model and solution approach

Our model is a generalization of the standard set covering model, which is often used for solving nurse scheduling problems. Let  $J$  be the set of feasible roster lines  $j$  and  $I$  be the set of demand periods  $i$ . An individual's *roster line* can be viewed as a sequence of *days on* and *days off*, where each day on contains a single *shift* identified by a label such as 'day', 'evening' or 'night'. Let  $a_{ij}$  be 1 if roster line  $j$  contains an active shift during period  $i$  and 0 otherwise. The general integer decision variable  $x_j$ ,  $\forall j \in J$  indicates the number of individual nurses which are scheduled under roster line  $j$ . Instead of assuming the demand values fixed, we consider them to be dependent on the number and type of patients undergoing surgery in the hospital at each moment. Hence, the cyclical surgery schedule as a whole determines the precise distribution of the workload over time. Let  $K$  denote the set of possible workload patterns that could be generated by modifying the surgery schedule. These will be obtained by enumerating all possible ways of assigning operating blocks to the different surgeons, subject to surgery demand and capacity restrictions. Each workload pattern  $k$  is described by a number of periodic demands  $d_{ik}$   $\forall i \in I$ . Let  $z_k$  be 1 if the surgery schedule that corresponds to workload  $k$  is chosen and 0 otherwise. Then, the problem can be stated as follows:

$$\text{Minimize } \sum_{j \in J} x_j \quad (1)$$

Subject to:

$$\sum_{j \in J} a_{ij} x_j \geq \sum_{k \in K} d_{ik} z_k \quad \forall i \in I \quad (2)$$

$$\sum_{k \in K} z_k = 1 \quad (3)$$

$$x_j \in \mathbb{N}^+ \quad \forall j \in J \text{ and } z_k \in \{0, 1\} \quad \forall k \in K \quad (4)$$

This model is very similar to models for integrating the scheduling of project tasks and employees (Alfares and Bailey, 1997, Alfares et al., 1999). It is solved by first solving the linear programming relaxation and then using a branching scheme to drive the solution into integrality. As the number of possible roster lines and workload patterns is usually too large to allow complete, a-priori enumeration, column generation is applied to solve the LP relaxation (see e.g. Barnhart et al., 1998, Caprara et al., 2003, Bard and Purnomo, 2005a, Bard and Purnomo, 2005b, Mehrotra et al., 2000, Jaumard et al., 1998, Mehrotra et al., 2000, Mason and Smith, 1998). Hereby, we distinguish between two types of pricing problems: a shortest path problem for generating a new roster line and an integer program for generating a new surgery schedule with resulting workload pattern (see e.g. Beliën and Demeulemeester, 2004). Extensive computational results on simulated data indicate that this integrated approach could lead to considerable savings in staffing costs and/or a more efficient use of available human resources.

**Keywords:** Rostering, surgery scheduling, column generation, integer programming.

## References

- Alfares, H. and Bailey, J. (1997). Integrated project task and manpower scheduling. *IIE Transactions*, 29:711–718.
- Alfares, H., Bailey, J., and Lin, W. (1999). Integrating project operations and personnel scheduling with multiple labor classes. *Production Planning & Control*, 10:570–578.
- Bard, J. F. and Purnomo, H. W. (2005a). A column generation-based approach to solve the preference scheduling problem for nurses with downgrading. *Socio-Economic Planning Sciences*, 39:193–213.
- Bard, J. F. and Purnomo, H. W. (2005b). Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164:510–534.
- Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329.
- Beliën, J. and Demeulemeester, E. (2004). Integer programming for building robust surgery schedules. Research Report OR 0446, Katholieke Universiteit Leuven, Department of Applied Economics.
- Caprara, A., Monaci, M., and Toth, P. (2003). Models and algorithms for a staff scheduling problem. *Mathematical Programming*, 98:445–476.
- Jaumard, B., Semet, F., and Vovor, T. (1998). A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, 107:1–18.
- Mason, A. J. and Smith, M. C. (1998). A nested column generator for solving rostering problems with integer programming. In *International Conference on Optimisation: Techniques and Applications*, pages 827–834.
- Mehrotra, A., Murphy, K. E., and Trick, M. A. (2000). Optimal shift scheduling: A branch-and-price approach. *Naval Research Logistics*, 47:185–200.

# AN ELECTROMAGNETIC META-HEURISTIC FOR THE NURSE SCHEDULING PROBLEM

Broos Maenhout<sup>1</sup> and Mario Vanhoucke<sup>1,2</sup>

<sup>1</sup>*Faculty of Economics and Business Administration, Ghent University, Gent, Belgium*

<sup>2</sup>*Operations & Technology Management Centre, Vlerick Leuven Gent Management School, Gent, Belgium*

*broos.maenhout@ugent.be • [mario.vanhoucke@ugent.be](mailto:mario.vanhoucke@ugent.be)*

Key words: Rostering, Heuristic search, Local search

## 1. INTRODUCTION

The nurse scheduling problem (NSP) assigns nurses to shifts per day taking both hard and soft constraints (e.g. hospital and departmental policies, labor laws, etc...) into account. The NSP must resolve conflicting viewpoints of the hospital and the nursing personnel. Hospitals are required to provide some minimum level of care during each shift (i.e. adequate coverage), while nurses want individualized high-quality schedules that take into account requests for days off, the exclusion of undesirable work patterns, and other personal considerations. The objective maximizes the preferences of the nurses and minimizes the total penalty cost from violations of the soft constraints. This is known as preference scheduling in nurse scheduling (Warner, 1976; Burke et al., 2004).

In our paper, a new meta-heuristic is presented for the NSP based on the electromagnetism framework of Birbil and Fang (2003). This so-called electromagnetism (EM) optimization heuristic for global optimization problems with bounded variables is a population-based meta-heuristic that operates in a multi-dimensional solution space where each solution point is moved by forces that obey the rules of electromagnetism. Attraction or repulsion is exerted on each solution point by all other points, which together compose the total force by which each solution point is moved. This heuristic is guided by the superposition principle which states that the force exerted on a point is inversely proportional to the distance between the points and directly proportional to the product of their charges. The underlying philosophy of this approach is that inferior solution points will prevent a move in their direction by repelling other solution points in the population and that attractive points will facilitate moves in their direction.

The purpose of our procedure is to present a generic framework for the nurse scheduling problem which can be applied to a wide range of hospital-specific cases.

## 2. DESCRIPTION OF THE PROCEDURE

We have adapted the basic EM algorithm by applying the nurse-shift pattern view (Cheang, 2003; Aickelin, 2000). In this view the decision variables represent shift patterns over a complete scheduling period and hence, implicitly incorporate many (case-specific) constraints. In this way, the different case specific constraints are handled as hard constraints and each nurse can have only a limited number of possible shift patterns. Some constraints, however, are incorporated in the objective function of the problem since it is impossible or explicitly not the purpose to model these constraints as hard constraints.

Our basic enhancements of the EM approach to the NSP assigns a (discrete) value (i.e. a shift pattern) to each nurse without taking the minimal coverage requirements into account. Since this solution approach often fails to find feasible solutions (i.e. the minimal coverage constraints), we follow the suggestions proposed by Beasley and Chu (1996) to deal with infeasible solutions, i.e. penalty functions and repair functions. A penalty function penalises infeasible solutions, whereas a repair function is a heuristic operator that transforms infeasible solutions into feasible solutions. We use a repair function which biases the move in the EM meta-heuristic to improve infeasibilities of the minimal coverage constraint. More precisely, rather than simply performing a move based on the calculated forces from a shift-pattern to another shift-pattern, more shift-specific information of individual patterns is exploited. Hence, in a move, a pattern is guided towards patterns which incorporate shifts in which shortages exist.

The schedules generated by the EM meta-heuristic are further intensified by local search. In our approach, there is opted for a variable neighbourhood search which combines local search heuristics. According to Glover (1997), the idea of changing neighbourhoods helps to explore the search space of combinatorial problems more effectively. Based on problem specific local information schedules are searched to be improved in terms of both soft constraint violation and total preference cost. In our procedure, we have implemented three local search algorithms, each focusing on a different part of the nurse scheduling matrix, i.e. on a shift-pattern, on a single day and on the complete matrix.

## 3. COMPUTATIONAL RESULTS

We have tested our novel approach on a randomly generated testset under a controlled design consisting of up to 100 nurses. Moreover, we have solved a real-life testset taken from a Belgian hospital. Our computational experiments indicate that the new EM approach is capable to produce consistently good results for challenging instances of the NSP.

## REFERENCES

- Aickelin, U., and Dowsland, K.A., 2000, Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem, *Journal of Scheduling*, 3, 139-153
- Beasley, J.E., and Chu, P.C., 1996, A genetic algorithm for the set covering problem, *European Journal of Operational Research*, 94, 392-404
- Birbil, S.I., and Fang, S.C., 2003, An electro-magnetism-like mechanism for global optimization, *Journal of Global Optimization*, 25, 263-282
- Burke, E.K., De Causmacker, P., Vanden Berghe, G., and Van Landeghem, H., 2004, The state of the art of Nurse Rostering, *Journal of Scheduling*, 7, 441-499
- Cheang, B., Li, H., Lim, A., and Rodrigues, B., 2003, Nurse rostering problems – a bibliographic survey, *European Journal of Operational Research*, 151, 447-460
- Glover, F., and Laguna, M., 1997, Tabu Search, *Kluwer Academic Publishers*, Boston, MA, 1997
- Warner H.W. (1976), Scheduling Nursing Personnel According to Nursing Preference, *Operations Research*, 24, 842-856

# **Search Algorithms**

Full Papers

# AN INVESTIGATION OF VARIABLE NEIGHBOURHOOD SEARCH FOR UNIVERSITY COURSE TIMETABLING

SALWANI ABDULLAH<sup>1</sup>, EDMUND K. BURKE<sup>1</sup> AND BARRY MCCOLLUM<sup>2</sup>

<sup>1</sup>*Automated Scheduling, Optimization and Planning Research Group, School of Computer Science & Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom.*  
{sqa@cs.nott.ac.uk,ekb@cs.nott.ac.uk}

<sup>2</sup>*Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN, United Kingdom*  
{b.mccollum@qub.ac.uk}

**Abstract:** The university course timetabling problem consists, in essence, of assigning lectures to a specific timeslot and room. The goal is to satisfy as many soft constraints as possible while constructing a feasible schedule. In this paper, we present a variable neighbourhood search approach with an exponential monte carlo acceptance criteria. This heuristic search approach is based on random-descent local search. The solution returned after exploring a neighbourhood structure is accepted based on the exponential monte carlo criteria. The approach is tested over eleven established datasets. The results demonstrate that the variable neighbourhood search approach is able to produce solutions that are competitive with state-of-the-art techniques from the literature.

**Key words:** Course Timetabling, Variable Neighbourhood Search

## 1. INTRODUCTION

Local search meta-heuristics are some of the most popular approaches in solving challenging optimisation problems including course timetabling (see, for example, Burke *et al.* 2003, Socha *et al.* 2002 and Di Gaspero and Schaerf 2002). It is beyond the scope of this paper to review the extensive literature on course timetabling but the interested reader can consult the following survey / overview papers (de Werra 1985, Schaerf 1999, Bardadym 1996, Burke *et al.* 1997, Carter and Laporte 1998, Burke and Petrovic 2002, Petrovic and Burke 2004 and Burke *et al.* 2004a). The success of finding good solutions for these problems is determined by the technique itself and the neighbourhood structure employed during the search. Ahuja *et al.* (2000) in their paper highlighted the importance of the neighbourhood structure in the local or neighbourhood search. They said,

*“A critical issue in the design of a neighbourhood search approach is the choice of the neighbourhood structure that is the manner in which the neighbourhood is defined.”*

Some techniques in the literature, like simulated annealing and tabu search, generally use a single neighbourhood structure throughout the search and focus more on the parameters that affect the acceptance of the moves rather than the neighbourhood structure. Thompson and Dowsland (1996, 1998) discussed how the choice of the neighbourhood structure affects the quality of solutions obtained for examination timetabling. The objective of this paper is to analyse the importance of different neighbourhood structures for course timetabling problem. We present and discuss a Variable Neighbourhood Search (VNS) approach to the problem.

In this paper, we consider the same course timetabling problem as described in Rossi-Dario *et al.* (2002) and Socha *et al.*, (2002). The problem consists of:

- A set of  $N$  courses
- 45 timeslots (5 days with 9 timeslots each day)
- A set of  $R$  rooms
- A set of  $F$  room features
- A set of  $M$  students.

It deals with a set of courses  $e = \{e_1, \dots, e_N\}$  that need to be mapped to 45 timeslots subject to a variety of hard and soft constraints. Hard constraints should not be violated under any circumstances and we call a timetable while satisfies all such constraints a feasible solution. Socha *et al.* (2002) present the following hard constraints:

1. *No student can be assigned to more than one course at the same time.*
2. *The room should satisfy the features required by the course.*
3. *The number of students attending the course should be less than or equal to the capacity of the room.*
4. *No more than one course is allowed at a timeslot in each room.*

Socha *et al.* (2002) present the following soft constraints that are penalised equally.

1. *A student has a course scheduled in the last timeslot of the day.*

2. *A student has more than 2 consecutive courses.*
3. *A student has a single course on a day.*

The objective of this problem is to minimize the number of soft constraint violations (i.e. a penalty cost) in a feasible solution.

This paper is organised as follows: the basic algorithm of VNS is discussed in the next section. The implementation of VNS to course timetabling is discussed in Section 3 followed by a discussion on the experiments and results in Section 4. Brief conclusions are drawn in Section 5.

## 2. VARIABLE NEIGHBOURHOOD SEARCH (VNS)

VNS was introduced by Mladenović and Hansen (1997). It is based upon the strategy of using more than one neighbourhood structure and of changing those structures systematically during the local search. This helps the VNS to explore neighbourhoods which are distant from the current solution and to jump to new solutions.

The basic VNS is made up of three stages: *shaking*, *local search* and *move*. More details can be found in Hansen and Mladenović (2001). A local search is applied repeatedly to obtain the local optima from the current solution (Mladenović and Hansen, 1997). Originally the basic VNS approach was a descent method. It does not accept a worsening solution to get out of local optima since the neighbourhood structures are varied regularly. Since the local optima in one neighbourhood structure is not necessarily a local optima in another neighbourhood structure, the change of the neighbourhood structures can be undertaken during the search. The termination criteria may be selected as a maximum number of iterations, the CPU time or a certain number of iterations without improvement.

Recently, research interest in the VNS approach has increased. For example Burke *et al.* (2004b) investigated VNS for nurse rostering problems where they employed two different search algorithms (steepest-descent and tabu search) implemented on a set of defined neighbourhood structures. Avanthay *et al.* (2003) implemented an adaptation of VNS for the graph colouring problem using the Tabucol algorithm (Hertz and Werra, 1987) as a local search employed on three neighbourhood structures (vertex, class and non-increasing neighbourhood). Other related works on VNS applied on a number of different problems can be found in Moreno Pérez *et al.* (2003), Mladenović *et al.* (2003), Hansen *et al.* (2001) and Caporossi and Hansen (2000).

## 3. VNS FOR COURSE TIMETABLING

This section discusses the applications of the basic VNS (referred to hereafter as VNS-Basic) and the modified VNS with exponential monte carlo acceptance criteria at the VNS level (referred to hereafter as VNS-EMC). The motivation for applying a monte carlo acceptance criteria is to better explore the search space.

### 3.1 Initial Solution: Constructive Heuristic

The initial solution is produced using a constructive heuristic which starts from an empty timetable. This feasible solution is obtained by adding or removing appropriate events (courses) from the schedule based on room availability (we attempt to schedule those courses with the least room



availabilities earlier on in the process), without taking into account any of the soft constraints, until the hard constraints are met. This constructive heuristic behaves like a saturation degree heuristic (see Carter and Laporte, 1996). The schedule is made feasible before starting the algorithms.

### 3.2 Local Search

A local search heuristic explores the neighbourhood at the present solution by iteratively performing local changes in order to improve the quality of a solution until a local optima (the best solution(s) in the defined neighbourhood) is found. In this paper, we develop a random-descent local search that only accepts an improved solution after exploring the nearest neighbours in the defined neighbourhood structure.

### 3.3 Acceptance Criteria for VNS

In this work, we use two acceptance criteria to be applied at the VNS level. The motivation for applying acceptance criteria at the VNS level is to jump to other distant solution points by accepting worse solutions (i.e. case (b)).

- a) Descent: It only accepts an improved solution.
- b) Exponential monte carlo: This is quite similar to the acceptance criteria of simulated annealing but we do not need a cooling schedule. Exponential monte carlo is only based on the solution quality (no tuning parameters are needed). It accepts a worse solution with a certain probability. Let us denote the old and new solutions by  $s$  and  $s'$ , respectively. The quality of the old and new solutions can be represented as  $f(s)$  and  $f(s')$ , respectively. The new solution  $s'$  is accepted if a generated random number  $[0,1]$  is less than  $e^{-\delta}$  where  $\delta = f(s') - f(s)$ . Increasing the value of  $\delta$  will decrease the probability of accepting a worse solution. More details on the exponential monte carlo can be found in Ayob and Kendall (2003).

### 3.4 Neighbourhood Structures within VNS

We implemented the following neighbourhood structures to be used at the local search level:

- (1) Move timeslot. Take 2 timeslots (selected at random), say  $t_i$  and  $t_j$  (where  $j > i$ ) and the timeslots are ordered  $t_1, t_2, \dots, t_{45}$ . Take all the exams in  $t_i$  and allocate them to  $t_j$ . Now take the exams that were in  $t_j$  and allocate them to  $t_{j-1}$ . Then allocate those that were in  $t_{j-1}$  to  $t_{j-2}$  and so on until we allocate those that were in  $t_{i+1}$  to  $t_i$  and terminate the process.
- (2) Move the highest penalty course (i.e. a course with the highest number of soft constraint violations). Take 10% of the courses at random. Then select the one with the highest penalty cost and allocate it to the timeslot which generates the lowest penalty and which does not create an infeasibility.
- (3) Move the highest penalty course from a random 30% selection of the courses to a random feasible timeslot.
- (4) Move the highest penalty course from a random 20% selection of the courses to a random feasible timeslot.

- (5) Move highest penalty course from a random 10% selection of the courses to a random feasible timeslot.
- (6) Select a course at random and find another course at random which can swap timeslots.
- (7) Choose single course at random and move to another random feasible timeslot.
- (8) Move 2 courses to random feasible timeslots.
- (9) Move 3 courses (as in (8)).
- (10) Move 4 courses (as in (8)).
- (11) Move 5 courses (as in (8)).
- (12) Select two timeslots at random and simply swap all courses in one timeslot with all courses in the other timeslot.

The neighbourhood structure described in (1) is implemented before we apply the VNS approach. This neighbourhood structure allows courses to be moved relative to each other except for courses that are scheduled in the same timeslot.

### 3.5 The VNS Algorithm

For the approach presented in this paper, we apply the set of neighbourhood structures as presented in subsection 3.3. Hard constraints are never violated during the course of the timetabling process.

Let  $n_k$  (where  $k = 1, \dots, K$ ) be a set of predefined neighbourhood structures. Note that  $K$  is the total number of neighbourhood structures to be used in the search. Let  $f(s)$  be the quality of the solution  $s$ . The local search starts by randomly generating a solution  $s'$  from the  $k^{th}$  neighbourhood. Starting from the initial solution  $s'$ , the local search sequentially visits the  $k^{th}$  neighbourhood of  $s'$  until a local optima  $s''$  is obtained. The solution  $s''$  is accepted if  $f(s'')$  is better than  $f(s)$ . This algorithm will also accept a worse solution with a certain probability that is generated from the exponential monte carlo acceptance criteria. Whenever a neighbourhood structure generates a better solution (or an accepted worse solution), the search starts over from the first neighbourhood. Otherwise, the next neighbourhood is employed. Based on our initial tests with the neighbourhood ordering for the course timetabling problem, we note that the best sequence of the neighbourhood structures is to order them in an increasing size. An increasing size-based neighbourhood structure ordering has also been experimented with in Burke et al. (2004b) where the authors claim that this ordering represents a strong approach in VNS when applied to nurse rostering problem. Figure 1 shows a schematic overview of the approach.

```

Initialisation:
(1) Select the set of neighbourhood structure  $n_k$ ,  $k=1, \dots, K$ 
    that will be used in the random descent local search;
    find the initial solution  $s$ ; choose a termination
    criteria;
(2) Record the best solution  $s_{best} \leftarrow s$  and  $f(s_{best}) \leftarrow f(s)$ ;
Repeat until the termination criteria is met:
(1) Set  $k \leftarrow 1$ ;
(2) Until  $k = K$ , repeat:
(a) Shaking: Generate a point  $s'$  at random from the  $k^{th}$ 
    neighbourhood of  $s$  ( $s' \in n_k(s)$ );
(b) Local search: Apply a random-descent local search with
     $s'$  as an initial solution until local optima  $s''$  is
    obtained.
(c) Move or not: Accept  $s''$  ( $s \leftarrow s''$ ) if it is better than
    incumbent solution  $s$ . Set  $s_{best} \leftarrow s$  and continue the
    search with  $n_1$ ;
    otherwise,
        if  $f(s'')$  is accepted by the acceptance criteria,
        then
             $s \leftarrow s''$ ;
            Continue the search with  $n_1$ ;
        otherwise
            Set  $k \leftarrow k+1$ 

```

Figure 1: Pseudo code of a modified VNS for course timetabling problem

We further investigate the performance of VNS by adding a tabu list (Glover and Laguna 1993, 1997), to penalize neighbourhood structures that are not performed. We apply a tabu restriction where the neighbourhood structure will be tabu active if the new solution obtained is greater than the old solution and it is rejected by the exponential monte carlo acceptance criteria. The basic idea of the tabu list is to prevent a neighbourhood structure that did not perform well recently from being chosen in the next evaluations, so that we can direct the search to other possible areas of the search space. Based on our preliminary tests, the tabu tenure is set to be 2. A higher value of the tabu tenure is not appropriate in our case because we keep neighbourhood structures in the tabu list instead of moves. The higher the value of the tabu tenure, the longer the neighbourhood structures remain tabu, which will limit the number of the available neighbourhood structures to be used in the next evaluation in the VNS approach. The neighbourhood structures will remain tabu active for a number of evaluations which is equal to a tabu tenure. We call this approach VNS-Tabu. The pseudo code for VNS-Tabu is given in Figure 2.

```

Initialisation:
(1) Select the set of neighbourhood structure  $n_k$ ,  $k=1,\dots,K$ 
    that will be used in the random descent local search;
    find the initial solution  $s$ ; choose a termination
    criteria;
(2) Record the best solution  $s_{\text{best}} \leftarrow s$  and  $f(s_{\text{best}}) \leftarrow f(s)$ ;

Repeat until the termination criteria is met:
(1) Set  $k \leftarrow 1$ ;
(2) Until  $k = K$ , repeat:
(a) Shaking: Generate a point  $s'$  at random from the  $k^{\text{th}}$ 
    neighbourhood of  $s$  ( $s' \in n_k(s)$ );
(b) Local search: Apply a random-descent local search with
     $s'$  as an initial solution until local optima  $s''$  is
    obtained.
(c) Move or not:
    if  $f(s'')$  is better then incumbent solution  $s$  or  $f(s'')$ 
    is accepted by the acceptance criteria, then
         $s \leftarrow s''$ 
        set  $k \leftarrow 1$ ;
        while  $k$  is in the tabulist
             $k \leftarrow k+1$ ;
        continue the search with  $n_k$ ;
    otherwise
        insert  $k$  to the tabulist;
        set  $k \leftarrow k+1$ ;
        while  $k$  is in the tabulist
             $k \leftarrow k+1$ ;

```

Figure 2: The pseudo code for VNS-Tabu

## 4 EXPERIMENTS AND RESULTS

The proposed method was tested on benchmark course timetabling problems from Socha *et al.* (2002) that are grouped into 5 small ( $N = 100$ ,  $R = 5$ ,  $F = 5$  and  $M = 80$ ) and 5 medium ( $N = 400$ ,  $R = 10$ ,  $F = 10$  and  $M = 200$ ) problems. We did not apply our VNS approach to the *large* dataset ( $N = 400$ ,  $R = 10$ ,  $F = 10$  and  $M = 400$ ) from Socha *et al.* (2002) since our constructive heuristic is unable to produce a feasible solution for this dataset. In Table 1, the test results on the benchmark course timetabling problems are presented for two types of variable neighbourhood search (i.e. VNS-Basic and VNS-EMC) “with and without ordering” of the neighbourhood structure. For the case of “without ordering”, the VNS will continue the search with the current neighbourhood if it yields an improvement rather than going back to the first (set  $k = 1$ ) neighbourhood structure (as in “with ordering” VNS) each time an improvement is found (or if the worse solution is accepted by the acceptance criteria). For instance, let  $k = 5$  represent the current neighbourhood. The local optima obtained from this neighbourhood structure will be compared to the incumbent solution. If there is an improvement in the solution, then we repeat the search at  $k = 5$ . Otherwise we move to the next neighbourhood structure (represented by  $k = 6$ ). This allows the next neighbourhood to be simply considered in the next search without needing to tune the sequence of the neighbourhood to determine

which neighbourhood will be employed. The motivation of this comparison is to see whether the sequence of neighbourhood structures plays a role in the VNS approach. We also make a comparison between the two VNS approaches to investigate the importance of accepting a worse move in order to jump to another part of the search space to obtain a better solution.

The approaches are coded in Microsoft Visual C++ version 6 under Windows. All experiments were run on an Athlon machine with a 1.2GHz processor and 256 MB RAM running under Microsoft Windows 2000 version 5. The number of evaluations for our approaches is 200000 (as in Socha *et al.*, 2002) and the best results out of 5 runs obtained are presented.

Table 1: Results on course timetabling problem

| Dataset        | Initial Solution | With ordering VNS |         | Without ordering VNS |         |
|----------------|------------------|-------------------|---------|----------------------|---------|
|                |                  | VNS-Basic         | VNS-EMC | VNS-Basic            | VNS-EMC |
| <i>small1</i>  | 261              | 8                 | 0       | 19                   | 1       |
| <i>small2</i>  | 245              | 12                | 0       | 7                    | 1       |
| <i>small3</i>  | 232              | 8                 | 0       | 11                   | 1       |
| <i>small4</i>  | 158              | 15                | 0       | 19                   | 0       |
| <i>small5</i>  | 421              | 5                 | 0       | 5                    | 0       |
| <i>medium1</i> | 914              | 418               | 338     | 445                  | 347     |
| <i>medium2</i> | 878              | 414               | 337     | 413                  | 354     |
| <i>medium3</i> | 941              | 441               | 384     | 462                  | 390     |
| <i>medium4</i> | 865              | 381               | 299     | 406                  | 324     |
| <i>medium5</i> | 780              | 390               | 307     | 416                  | 315     |
| <i>large</i>   | 100%Inf.         | -                 | -       | -                    | -       |

From Table 1, we can see that the “with ordering” VNS is better than or equal to the “without ordering” VNS for both VNS-Basic and VNS-EMC (except for *small2* and *medium2* datasets for VNS-Basic which is slightly higher). This shows that the ordering of the neighbourhood structure in the VNS approach and which neighbourhood the search will initiate after obtaining a better solution (or a worse accepted solution) is important. We believe this is because we give a chance for the search approach to explore by using different neighbourhood structures which can lead to different search spaces rather than letting the search approach employ the same neighbourhood structure until no more improvement on the solution quality is obtained (i.e. where the next neighbourhood structure only be employed on an unimproved solution). The results obtained from VNS-EMC are better than VNS-Basic for both the “with and without ordering” cases. This shows that accepting worse solutions (as a diversification strategy) helps the VNS approach to better explore the search space as suggested by Glover and Laguna (1997). Figures 3a and b show the behaviour of our approach applied to the *medium2* and *medium3* datasets on “with and without ordering” for VNS-EMC (to show the importance of ordering). Figures 4a and b show the behaviour of the VNS-Basic and VNS-EMC methods (for the case of “with ordering” VNS) on the *small4* and *small5* problems which indicates the importance of accepting worse solution in order to obtain a better final solution.

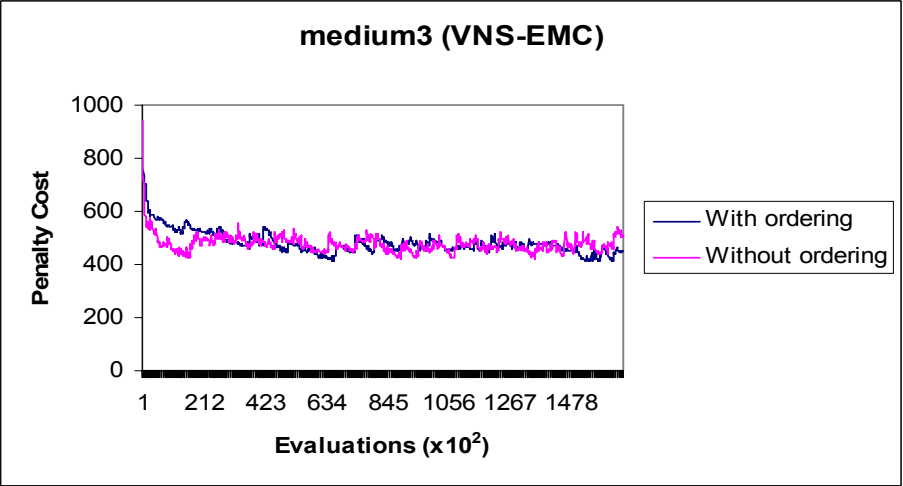
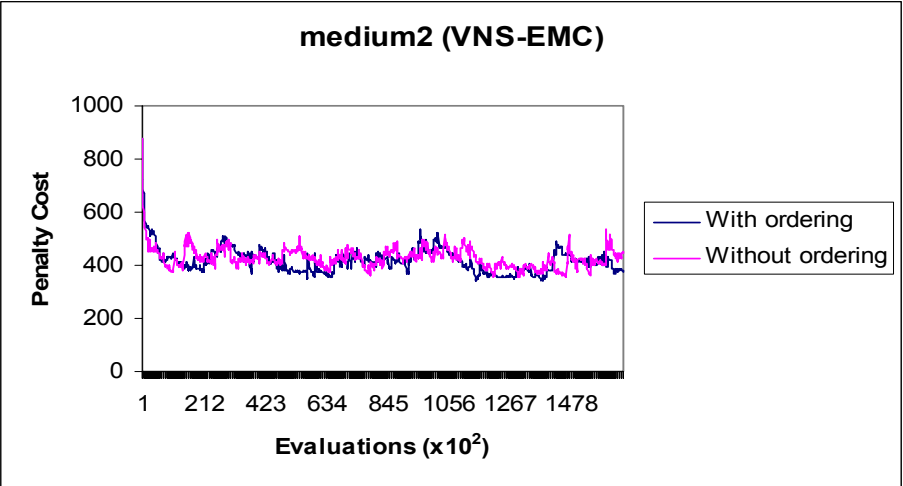
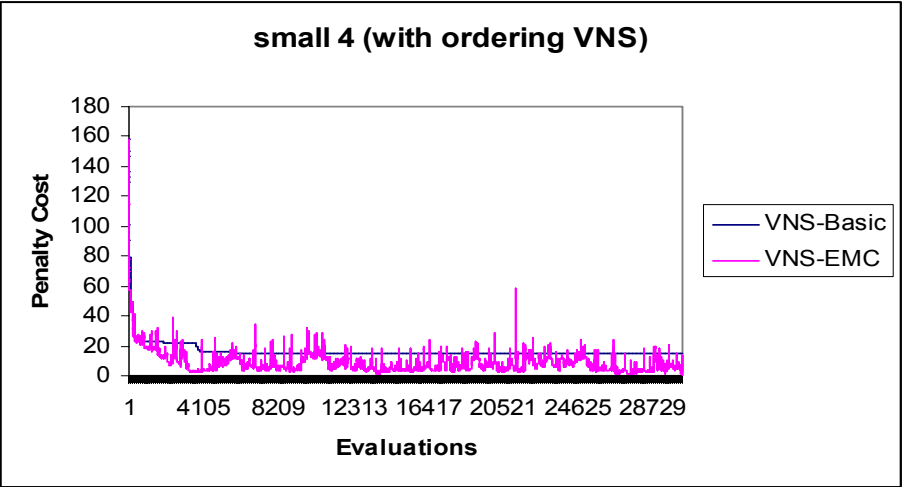


Figure 3a and b: The behaviour of *medium2* and *medium3* datasets, respectively



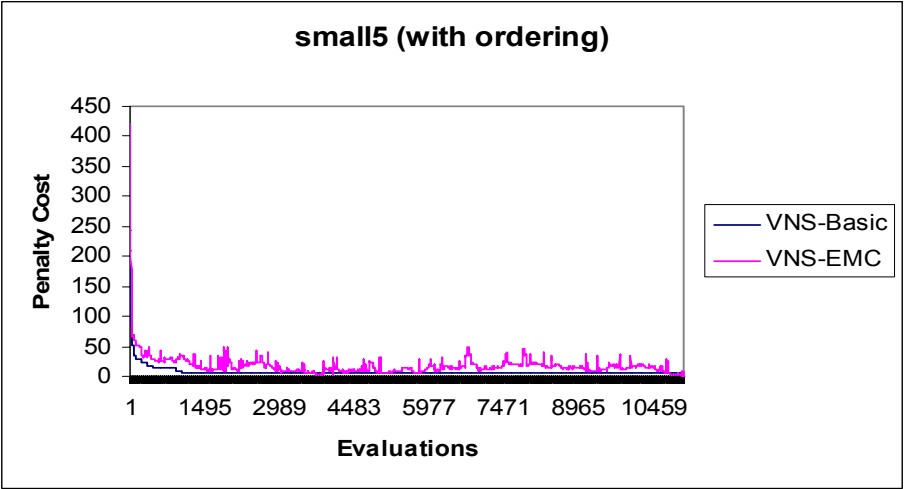


Figure 4a and b: The behaviour of *small4* and *small5* datasets, respectively

In all the figures, the x-axis represents the number of evaluations while the y-axis represents the penalty cost. Every point in the graphs corresponds to the penalty cost and number of evaluations of a separate solution. These graphs show how our algorithm explores the search space. In Figures 3a and b (and also for Figures 4a and b for VNS-EMC), the curves move up and down because we accept worse solutions with a certain probability in order to escape from local optima. The analysis of the graphs in Figures 3a and b show that the slope of the curves is relatively steep which indicates the high improvement in the quality of the solutions at the beginning of the search for both the “with and without ordering” cases of VNS-EMC. The improvement of the solution becomes slower as the search time increases but, with a good sequence of neighbourhood structures, the “with-ordering VNS” is able to find the better solution compared to the “without ordering VNS”. This shows the importance of the sequence of neighbourhood. The graph in Figures 4a and b also illustrates the same behaviour as in Figures 3a and b at the early stage of the search with a high improvement in the solution quality. However, VNS-Basic gets stuck in the local optima after a certain number of evaluations. Whilst, by accepting worse solutions, the VNS-EMC is able to make further improvements and obtain a better solution (i.e. zero penalty cost). This indicates how a diversification strategy (i.e. accepting worse solutions) efficiently helps the technique to explore the search space.

Table 2 shows the comparison of our approach with other available approaches in the literature: A local search method and ant algorithm by Socha *et al.* (2002); a tabu-search hyperheuristic and a graph hyperheuristic by Burke *et al.* (2003) and Burke *et al.* (2005), respectively. Socha *et al.* (2002) present the average results out of 50 runs on the *small* problems, 40 runs on the *medium* problems and 10 runs on the *large* problem for their local search and ant algorithm. The term “x%Inf” in Table 2 indicates a percentage of runs that failed to obtain feasible solutions. The number of evaluations for both local search and the ant algorithm by Socha *et al.* (2002) was approximately 200000. The number of evaluations for the tabu-based hyperheuristic by Burke *et al.* (2003) was 12000 for the *small* problems, 1200 for the *medium* problems and 5400 for the *large* problem. Burke *et al.* (2005) did not note the number of evaluations in their approach.

Table 2: Comparison results on course timetabling problem

| Dataset        | VNS-<br>EMC | Local search<br>(Socha <i>et al.</i> ,<br>2002) | Ant<br>Algorithm<br>(Socha <i>et al.</i> ,<br>2002) | Tabu-based<br>hyperheuristic<br>(Burke <i>et al.</i> ,<br>2003) | Graph<br>hyperheuristic<br>(Burke <i>et al.</i> ,<br>2005) |
|----------------|-------------|-------------------------------------------------|-----------------------------------------------------|-----------------------------------------------------------------|------------------------------------------------------------|
| <i>small1</i>  | <b>0</b>    | 8                                               | 1                                                   | 1                                                               | 6                                                          |
| <i>small2</i>  | <b>0</b>    | 11                                              | 3                                                   | 2                                                               | 7                                                          |
| <i>small3</i>  | <b>0</b>    | 8                                               | 1                                                   | <b>0</b>                                                        | 3                                                          |
| <i>small4</i>  | <b>0</b>    | 7                                               | 1                                                   | 1                                                               | 3                                                          |
| <i>small5</i>  | <b>0</b>    | 5                                               | <b>0</b>                                            | <b>0</b>                                                        | 4                                                          |
| <i>medium1</i> | 338         | 199                                             | 195                                                 | <b>146</b>                                                      | 372                                                        |
| <i>medium2</i> | 326         | 202.5                                           | 184                                                 | <b>173</b>                                                      | 419                                                        |
| <i>medium3</i> | 384         | 77.5% Inf                                       | <b>248</b>                                          | 267                                                             | 359                                                        |
| <i>medium4</i> | 299         | 177.5                                           | <b>164.5</b>                                        | 169                                                             | 348                                                        |
| <i>medium5</i> | 307         | 100% Inf                                        | 219.5                                               | 303                                                             | <b>171</b>                                                 |
| <i>large</i>   | 100%Inf     | 100% Inf                                        | <b>851.5</b>                                        | 80% Inf 1166                                                    | 1068                                                       |

The best results are presented in bold. In terms of feasibility, we can see that our approach was able to produce ten out of eleven feasible solutions, whereas local search only produced nine feasible solutions (with two infeasible solutions for the *medium4* and *large* datasets). Other approaches are able to produce feasible solutions for all datasets. It can be seen that our approach produces better or equivalent results on *small* datasets when compared against all the other methods. Indeed, it is the only approach which can get zero penalty solutions on all 5 *small* problems. For the *medium* datasets, our results are comparable to the other results published in the literature except on the *medium3* and *medium5* datasets, where our method performs quite poorly. We believe that our approach performs well for *small* datasets because we might have more feasible solution points in our search space compared to *medium* datasets. We believe that we should be more selective in choosing the neighbourhood structures while solving the *medium* datasets.

Table 3 shows the comparison of the results obtained from VNS-Tabu, VNS-EMC and the best known results.

Table 3: The results on VNS-Tabu

| Dataset        | VNS-Tabu | VNS-EMC  | Best known   |
|----------------|----------|----------|--------------|
| <i>small1</i>  | <b>0</b> | <b>0</b> | 1            |
| <i>small2</i>  | <b>0</b> | <b>0</b> | 2            |
| <i>small3</i>  | <b>0</b> | <b>0</b> | <b>0</b>     |
| <i>small4</i>  | <b>0</b> | <b>0</b> | 1            |
| <i>small5</i>  | <b>0</b> | <b>0</b> | <b>0</b>     |
| <i>medium1</i> | 317      | 338      | <b>146</b>   |
| <i>medium2</i> | 313      | 326      | <b>173</b>   |
| <i>medium3</i> | 357      | 384      | <b>248</b>   |
| <i>medium4</i> | 247      | 299      | <b>164.5</b> |
| <i>medium5</i> | 292      | 307      | <b>171</b>   |
| <i>large</i>   | 100% Inf | 100%Inf  | <b>851.5</b> |

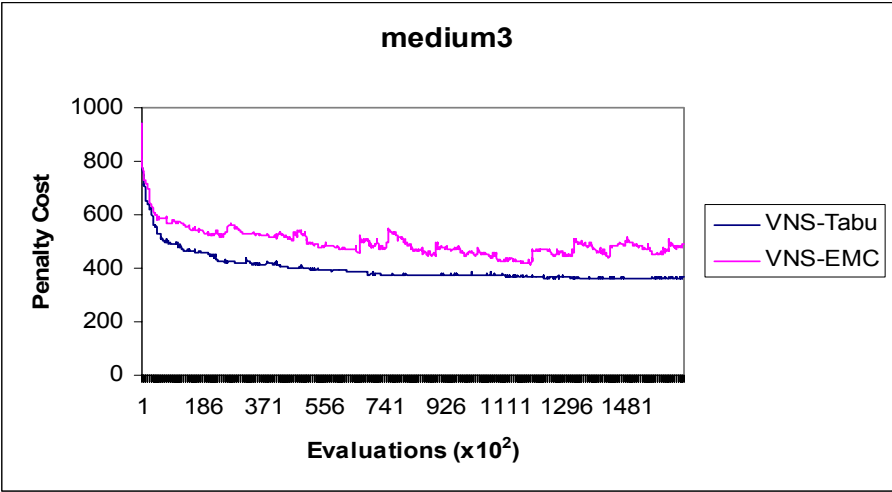


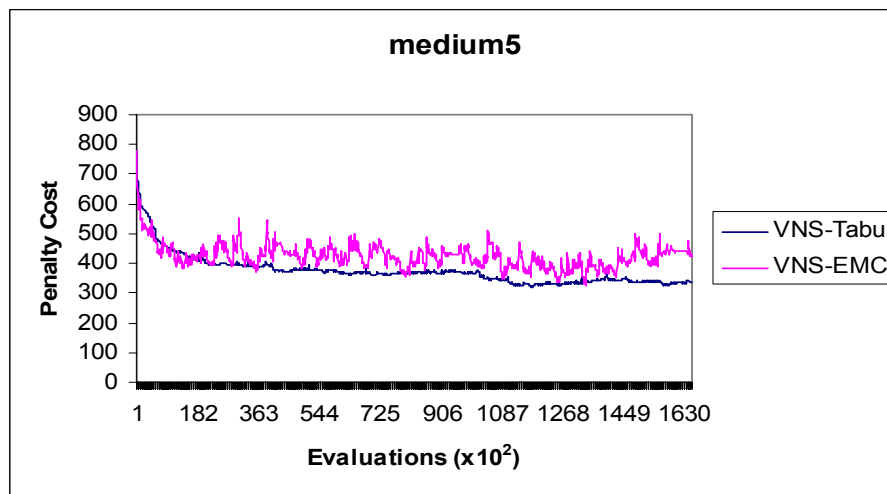
We are interested to compare the results from VNS-Tabu with our previous results to show that the tabu list can help to minimize the objective function for this problem. The results in Table 3 show the best result obtained from five runs on the VNS-Tabu. It can be seen that there is no difference between VNS-Tabu and VNS-EMC in terms of the penalty cost when applied to the *small* datasets. However, there are slight changes in the penalty cost for the *medium* datasets (as shown in italics). The percentage improvement ( $\Delta \%$ ) obtained by applying the VNS-Tabu compared to VNS-EMC for the *medium* dataset is computed as:

$$\Delta \% = (\text{best VNS-EMC} - \text{best VNS-Tabu}) * 100 / \text{best VNS-EMC}$$

This shows that the VNS-Tabu managed to reduce the penalty cost between 4.0% to 17.4%. Although we cannot beat the best known results for the *medium* datasets, we are able to repair our results especially for the *medium3* and *medium5* datasets. Figures 5a and b show the behaviour of the VNS-Tabu when applied to the *medium3* and *medium5* datasets.

We use the same representation as in Figures 3 and 4 where the x-axis represents the number of evaluations while the y-axis represents the penalty cost. They show that the penalty cost can be quickly reduced at the beginning of the search where there appears to be a lot of room for improvement and this becomes less pronounced towards the end of the search (as in Figures 3a and b). It can be seen from Figure 5a that the VNS-Tabu is better compared to VNS-EMC throughout the search. However, in Figure 5b, the VNS-EMC is slightly better than VNS-Tabu in the early part of the search. Note that, by prolonging the search process we can see that the pattern of the VNS-Tabu is similar for both *medium3* and *medium5* datasets which show that the VNS-Tabu is able to find a good solution if given more search time. We believe that with the help of the tabu list, the VNS-Tabu performs better than VNS-EMC and we are able to find a better solution because the neighbourhood structures that are not used will not be employed in the next evaluations (the algorithm will only be supplied with the neighbourhood structures that are currently used) unlike the VNS-EMC approach.





Figures 5a and b: The behaviour of VNS-Tabu applied to *medium3* and *medium5* datasets, respectively

## 5 CONCLUSIONS

The overall goal of this paper was to investigate a modified VNS approach for the course timetabling problem. Preliminary comparisons indicate that our VNS approach is competitive with other approaches in the literature and is particularly suitable for *small* problems. On the five (out of eleven) *small* benchmark problems, the VNS approach produced solutions that were better than or equal to those published results (obtaining zero penalty cost for all five problems). Our future work will try to intelligently select a subset of neighbourhood structures from a much larger pool of neighbourhoods.

## Acknowledgement

This work has been supported by the Public Services Department of Malaysia (JPA) and the University Kebangsaan Malaysia (UKM).

## References

- Ahuja, R.K., Orlin, J.B. and Sharma, D. (2000) Very large scale neighbourhood search. *International Transaction in Operational Research* 7, pp 301-317.
- Avanthay, C., Hertz, A. and Zufferey, N (2003) A variable neighbourhood search for graph colouring. *European Journal of Operational Research* 151, pp 379-388.
- Ayob, M. and Kendall, G. (2003) A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. *Proc. Of the International Conference on Intelligent Technologies, InTech'03*, Chiang Mai, Thailand, 17-19 Dec, pp 132-141.
- Bardadym, V. (1996) Computer-Aided School and University Timetabling: The New Wave. In Edmund Burke and Peter Ross, editors, *The Practice and Theory of Automated Timetabling I: Selected papers from the 1<sup>st</sup> International Conference*, Springer Lecture Notes in Computer Science Vol 1153, pp 22-45.

- Burke EK, Jackson KS, Kingston JH and Weare RF (1997) Automated Timetabling: The State of the Art. *The Computer Journal*, Vol. 40, No. 9, pp 565-571
- Burke EK and Petrovic S (2002) Recent Research Direction in Automated Timetabling. *European Journal of Operational Research* 140, pp 266-280.
- Burke, E.K., Kendall, G. and Soubeiga, E. (2003) A tabu search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9(6), pp 451-470.
- Burke EK, Kingston J and de Werra D (2004a) Applications to Timetabling. Handbook of Graph Theory, (editors, J. Gross and J. Yellen), Chapman Hall/CRC Press, pp 445-474.
- Burke, E. K., De Causmaecker, P. and Vanden Berghe, G.(2004b) *Novel metaheuristic Approaches to Nurse Rostering Problems in Belgian Hospitals*, Ch. 44 in J. Leung: Handbook of Scheduling: Algorithms, Models and Performance Analysis, CRC Press, pp 44.1-44.18.
- Burke, E.K., Meisels, A., Petrovic, S. and Qu, R. (2005) A graph-based hyper heuristic for timetabling problems. Accepted for publication in the *European Journal of Operational Research*.
- Caporossi, G. and Hansen, P. (2000) Variable neighbourhood search for external graphs: I The AutoGraphiX system. *Discrete Mathematics* 212, pp 29-44.
- Carter, M.W. and Laporte, G. (1996) Recent developments in practical examination timetabling. In Edmund Burke and Michael Carter, editors, *The Practice and Theory of Automated Timetabling I: Selected Papers from the 1<sup>st</sup> International Conference*, Springer Lecture Notes in Computer Science Vol 1153, pp 3-21.
- Carter, M.W. and Laporte, G. (1998) Recent developments in practical course timetabling. In Edmund Burke and Peter Ross, editors, *The Practice and Theory of Automated Timetabling II: Selected Papers from the 2<sup>nd</sup> International Conference*, Springer Lecture Notes in Computer Science Vol 1408, pp 3-19.
- Di Gaspero, L. and Schaerf, A. (2002) Multi-neighbourhood local search with application to course timetabling. In Edmund Burke and Patrick De Causmaecker, editors, *The Practice and Theory of Automated Timetabling IV: Selected Papers from the 4<sup>th</sup> International Conference*, Springer Lecture Notes in Computer Science Vol 2740, pp 262-275.
- de Werra D. (1985) An Introduction to Timetabling. *European Journal of Operations Research*, 19, pp 151-162.
- Glover and Laguna (1997) Tabu search. *Kluwer Academic Publisher*, ISBN 0-7923-8187-4.
- Glover, F and Laguna, M. (1993) Tabu Search. In Reeves, C.R. (eds.) *Modern Heuristic Technique for Combinatorial Problems*. Scientific Publications, Oxford.
- Hansen, P. and Mladenović, N. (2001) Variable neighbourhood search. *European Journal of Operational Research* 130, pp 449-467.
- Hansen, P., Mladenović, N. and Perez-Britos, D. (2001) Variable neighbourhood decomposition search, *Journal of Heuristics* 7, pp 335-350.
- Hertz, A. and de Werra, D. (1987) Tabu search techniques for graph colouring. *Computing* 39, pp 345-351.
- Mladenović, N. and Hansen, P. (1997) Variable neighbourhood search. *Computers and Operations Research* 24(11), pp 1097-1100.
- Mladenović, N., Petrović J., Kovačević-Vujčić, V. and Čangalović, M. (2003) Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research* 151, pp 389-399.
- Morena Pérez, J.A., Marcos Moreno-Vega, J. and Rodríguez Martín, I. (2003) Variable neighbourhood tabu search and its application to the median cycle problem, *European Journal of Operational Research* 151, pp 365-378.

- Petrovic S and Burke EK (2004) University Timetabling. Ch. 45 in the Handbook of Scheduling: Algorithms, Models, and Performance Analysis (eds. J. Leung), Chapman Hall/CRC Press.
- Rossi-Doria, O., Blum, C., Knowles, J., Samples, M., Socha, K. and Paechter, B. (2002) A local search for the timetabling. *Proceedings of the 4<sup>th</sup> International Conference on Practice and Theory of Automated Timetabling (PATAT)*, pp 124-127.
- Schaerf A (1999) A Survey of Automated Timetabling. *Artificial Intelligence Review* 13(2), pp 87-127.
- Socha, K., Knowles, J. and Samples, M. (2002) A max-min ant system for the university course timetabling problem. *Proceedings of the 3<sup>rd</sup> International Workshop on Ant Algorithms, ANTS 2002, Springer Lecture Notes in Computer Science* Vol 2463 (10), pp 1-13.
- Thompson, J. and Dowsland, K. (1996) General cooling schedule for a simulated annealing based timetabling system. In Burke, E.K. and Ross, P. (eds). *Practice and Theory of Automated Timetabling: Selected Papers from the 1<sup>st</sup> International Conference. Springer Lecture Notes in Computer Science* Vol 1153, pp 345-363.
- Thompson, J. and Dowsland, K. (1998) A robust simulated annealing based examination timetabling system. *Computers and Operations Research* 25, pp 637-648.

# A Zero-Inventory Production-Distribution Problem with Time-Window Constraints

R. Armstrong, L. Lei and S. Gao  
Rutgers Business School, Rutgers University

In this paper, we study a sub-problem of integrated production and distribution for a perishable product that cannot be inventoried. The manufacturing facility has a limited production rate, and the delivery truck has a non-negligible traveling time between locations. The order in which customers may receive deliveries is fixed. Each customer requests a delivery quantity and a time window for receiving the delivery. Since the production facility and the shipping trucks are limited resources, not all the customers may receive the delivery within their specified time windows. The problem is then to choose a subset of customers from the given sequence to receive the deliveries to maximize the amount of demand satisfied without violating the product lifetime, the production capacity, and the delivery time window constraints. The general version of the problem is NP-hard even with relaxed time windows and instantaneous truck traveling time. We analyze the properties of this problem and show the conditions under which optimality is achieved. A greedy heuristic is proposed based on this analysis. Empirical observations on the error gaps achieved by this heuristic are reported.

*(Key words: integrated production and distribution, time window constraints, zero-inventory, heuristics, empirical studies)*

## 1. Introduction

Many business applications involve make-to-order manufacturing and distribution operations, where customers accept deliveries only within specified time windows (Ghiani, et al. [1]). Scheduling the operations to meet such time window constraints becomes challenging when product inventory is not allowed. The manufacturing practice that motivated our study on this type of scheduling problem is the one encountered at a company that produces various adhesive chemicals. One of the products, an adhesive material used for making plywood panels, has a life span of 7 days, after that the strength of adhesiveness degrades quickly. During the house construction season, business customers place orders with the company and specify the time windows within which they expect to receive the delivery. Due to the nature of the product, no inventory is maintained and the production and distribution operation schedules must be highly coordinated. When the production facility has a limited production rate and when the transportation time is not instantaneous, any inefficiency in this integrated schedule may either cause the product to expire before it reaches the customers

(in which case the production resource is wasted and additional expenses will be incurred to provide an environmental safe disposal of expired products), or the delivery may fail to meet the customer's receiving time windows (in which case the contract with the customer is cancelled or a special arrangement must be made). There are many other examples of such short shelf-life (perishable) products. Radioactive material in biomedical research and clinic applications typically have a short shelf-life (e.g., 8 days for Iodine 131 and only 64 hours for Yttrium 90 (from [www.perkinelmer.com](http://www.perkinelmer.com))). Bakery products, with a shelf life of greater than seven days may suffer loss or significant reduction in marketability. A 36-hour delay in shipping farmland products would eliminate as high as 40% of their shelf-life (from [www.fda.gov](http://www.fda.gov)).

The resulting production and distribution problem (PDP) for short shelf-life products is concerned with *when and how much to produce and which truck route to follow so that a specified objective function (e.g., the total cost, the fleet size, the total time span, or the total amount of demand, etc) is optimized while all the constraints are satisfied*. When the production and transportation capacities are constrained and the customer receiving time windows are fixed, it is sometimes necessary to give up the orders by certain customers. Customers whose orders cannot be satisfied must be notified immediately so that they may seek another supplier or work out internally to allow a change in the delivery time window. Optimally solving such an integrated scheduling problem subject to the given constraints is usually difficult due to its inherent combinatorial nature.

In this paper, we consider a *subproblem* of this integrated production and distribution problem. We are interested in coordinating the operations of a production line and a single delivery truck with a given customer sequence which specifies the relative orders that customers may receive the deliveries. The given customer sequence is represented as  $S = \langle 1, 2, 3, \dots, n-1, n \rangle$ , where  $n$  denotes the total number of customers to be evaluated for receiving deliveries. By a given customer sequence, it means that for all  $i, j \in S$ ,  $i < j$ , if both customers  $i$  and  $j$  receive the deliveries in a distribution schedule, then  $i$  should receive the delivery before  $j$  does. Real life situations with a fixed customer sequence occur, for examples, when the customer orders are handled in a first come first serve basis or when the customers are geographically located along an interstate highway. Customer  $i$ ,  $i \in S$ , has demand  $d_i$  and a specified receiving time window  $[a_i, b_i]$  within which a delivery can be realized. That is, if customer  $i$  receives a delivery,

then whenever the truck arrives at  $i$  before  $a_i$ , a waiting time is incurred, and whenever the truck arrives after  $b_i$ , the delivery is rejected and the demand is lost. The plant that produces the product has a limited production rate  $r > 0$ , so that it takes  $d_i / r$  time units to produce for customer  $i$ . No partial order fulfillment is allowed. Items once produced must be delivered to the respective customer within  $B$  time units or the product will expire (i.e., the life span constraint). We assume the truck has enough capacity to carry all the supplies to the  $n$  customers in set  $S$ . Therefore, the truck capacity is not a concern.

Since both the manufacturing and transportation resources are limited and the time windows could be tight, there may be a subset of customers in set  $S$  that will not be served. The resulting problem is to choose a subset of customers,  $\sigma$ , where  $\sigma \subset S$ , to receive the deliveries so that the total amount of feasibly satisfied demand is maximized. We denote this problem as **P**. Note that for any given subset  $\sigma$ , the production and distribution schedule is known and fixed. Therefore, any feasible subset  $\sigma$  that satisfies the production capacity, the product life span, and the delivery time window constraints, defines a feasible solution/schedule for **P**.

**Example 1.** Suppose that the given sequence of customer orders is  $S = \langle A, B, C, D, E \rangle$ ,  $r=10$ , and the customer-specific data are listed in the table below:

| Customer $j$ | A  | B  | C  | D  | E  |
|--------------|----|----|----|----|----|
| $d_k$        | 30 | 40 | 60 | 30 | 40 |
| $a_k$        | 19 | 26 | 31 | 32 | 38 |
| $b_k$        | 24 | 32 | 36 | 35 | 39 |

Let  $\sigma=S$ , by which the delivery truck leaves the plant as soon as the order for customer E is completed. Since the product life span starts as soon as a customer order is completed, the product for customer B would expire before the truck arrives at the customer. In addition, the truck will arrive at customers D and E later than the latest delivery time. Therefore,  $\sigma$  is not a feasible schedule (see Figure 1a).

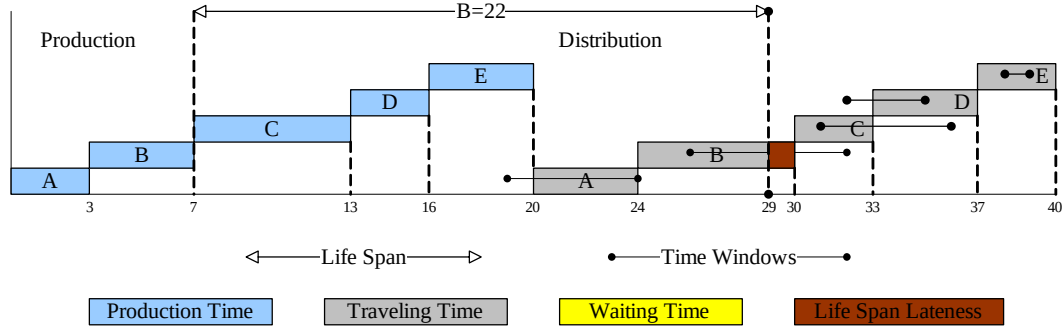


**Figure 1a.** An illustrative example of **P**.

One of the feasible solution/schedule to this example problem is achieved by deleting customer E, or let  $\sigma' = S \setminus \{E\} = \langle A, B, C, D \rangle$ , which satisfies all the constraints and results in a total amount of unsatisfied demand of 4 units. Note that with the new schedule  $\sigma'$ , there will be a waiting time at customers C (see Figure 1b). The optimal solution to this example problem is given by  $\sigma^* = S \setminus \{D\} = \langle A, B, C, E \rangle$ , which results in the minimum amount of unsatisfied demand of 3 units.

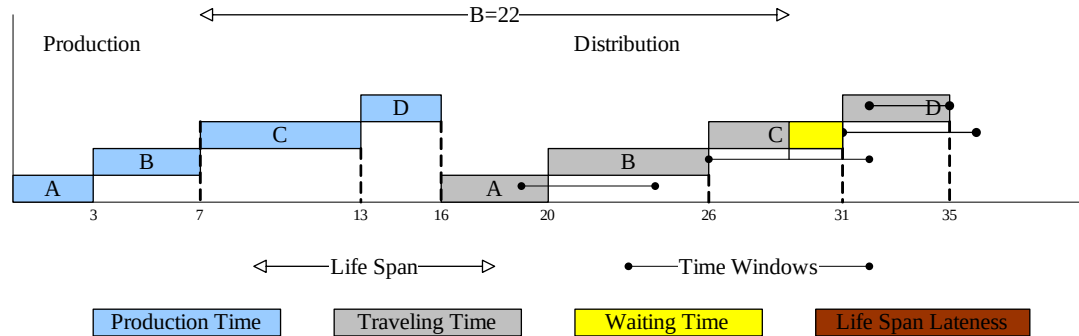


**Figure 1b.** A feasible schedule is obtained after customer E is deleted.

Identifying the optimal subset of customers to be deleted becomes difficult when  $n$  is large. In section 2, we show the general version of **P** is NP-hard, even though it becomes trivial when  $B = b_i = \infty, \forall i \in S$ .

In the literature, most published results on PDP can be classified into the two categories, those assuming negligible transportation cost but a non-negligible transportation time and those assuming an instantaneous transportation time but a



non-negligible cost. An early work on PDP with non-negligible transportation time was by Potts [2]. He considered the case with a single machine but sufficient number of vehicles so that once a job is completed it is transported to the customer without the need of consolidating the deliveries (i.e., one job per direct shipment trip). The objective was to sequence the jobs on that single machine so that the makespan (including the delivery time to the last customer) is minimized. A heuristic and worst case performance bound were discussed. Heuristics and approximation algorithms on extensions of this classical model were given by Hall and Shmoys [3], Woeginger [4], and Zdrzalka [5]. Lee and Chen [6] showed that most of these variations are NP-hard, but a few cases are (pseudo) polynomial time solvable. Another important subarea in PDP assumes a negligible shipping time but considers the transportation cost. A classical model of this (Pinedo [7]) is the single machine job sequencing problem with an objective to optimize a machine scheduling performance measure plus the transportation cost as a function of the number of delivery batches subject to a batch size. Complexity analysis and algorithms for different variations of this classical model were studied by Cheng and Gordon [8], Cheng, Gordon and Kovalyov [9], Cheng, Kovalyov and Lin [10], Herrmann and Lee [11], Yuan [12], and Wang and Cheng [13]. A more recent paper related to this area is given by Hall and Potts [14]. They considered a three-stage supply chain process with a supplier, manufacturer and customers. The PDP problems posted from a supplier's view, the manufacturer's view, and entire chain's view were discussed. Both special cases with polynomial algorithms and general case complexity analysis were presented. A comprehensive review on the related work can be found in Chen [15].

The general PDP that requires an integrated simultaneous optimization of production sequencing and transporter routing is even harder to solve. Very few related studies in this regard are available in the literature. Among those, Hurter and Van Buer [16], and Van Buer, Woodruff and Olson [17], proposed heuristics for coordinating the two type of schedules (production and distribution) for a problem encountered in news paper industry. Geismar, Lei and Sriskadarajah [18] studied an integrated optimization problem with a perishable product and the truck routing issues. The objective is to minimize the total time needed to produce and to deliver to a given set of customers (i.e., makespan) without the time window constraints. They proved that the problem in general is NP-hard in strong

sense, and proposed and evaluated a set of greedy heuristics against the lower bounds on the optimal solutions.

In section 2, we analyze the properties of  $\mathbf{P}$  and the conditions of optimality, upon which a greedy search algorithm is proposed in section 3. The empirical observations on the computational performance of this algorithm under randomly generated test cases are reported in section 4, and finally in section 5 we discuss future extensions from this study.

## 2. Properties of $\mathbf{P}$

We shall start our analysis with a formal definition of  $\mathbf{P}$ . Let

- $Y_{i,j} = 1$ : If the truck visits customer  $j$  immediately after visiting customer  $i$ ;  
 $w_j$ : Waiting time at customer  $j$  if  $Y_{ij} = 1$ , for some  $i < j$ , and if the truck arrives at customer  $j$  before  $a_j$ . Let indices 0 and  $n+1$  denote the plant, and let  $d_{n+1} = 0$ ;

Then,  $\mathbf{P}$  can be stated as the following mixed integer program.

$$\text{Max. } \sum_{i=0}^{n-1} \sum_{j=i+1}^n Y_{i,j} \cdot d_j \quad // \text{Total amount of satisfied demand} \quad (1)$$

Subject to:

*Time window constraints for delivery times*

$$\frac{1}{r} \sum_{i=0}^{n-1} \sum_{j=i+1}^n Y_{i,j} \cdot d_j + \sum_{i=0}^{k-1} \sum_{j=i+1}^k Y_{i,j} \cdot \tau_{i,j} + \sum_{i=1}^k w_i \geq a_k \cdot \sum_{i=0}^{k-1} Y_{i,k} \quad k = 1, 2, \dots, n \quad (2)$$

$$\begin{aligned} \frac{1}{r} \sum_{i=0}^{n-1} \sum_{j=i+1}^n Y_{i,j} \cdot d_j + \sum_{i=0}^{k-1} \sum_{j=i+1}^k Y_{i,j} \cdot \tau_{i,j} + \sum_{i=1}^k w_i \\ \leq b_k + M \cdot (1 - \sum_{i=0}^{k-1} Y_{i,k}) \quad k = 1, 2, \dots, n \quad (3) \end{aligned}$$

*Product life-span constraint*

$$\frac{1}{r} \sum_{i=k}^{n-1} \sum_{j=i+1}^n Y_{i,j} \cdot d_j + \sum_{i=0}^{k-1} \sum_{j=i+1}^k Y_{i,j} \cdot \tau_{i,j} + \sum_{i=1}^k w_i \leq B, \quad k = 1, 2, \dots, n \quad (4)$$

*Flow balance constraints*

$$\sum_{h=0}^{i-1} Y_{h,i} = \sum_{j=i+1}^{n+1} Y_{i,j} \quad i = 1, 2, \dots, n \quad (5)$$

$$\sum_{h=1}^n Y_{0,h} = 1, \quad \sum_{h=1}^n Y_{h,n+1} = 1 \quad (6)$$

$$Y_{i,j} = 0 \text{ or } 1, \quad w_i \geq 0, \quad \forall i, j, k \in S$$

While the relative order in which customers may be visited,  $S$ , is fixed, problem  $\mathbf{P}$  is non-trivial. To see this, consider a special case in which we completely relax the time windows, allow an instantaneous traveling time, and let  $d_1 > \sum_{j=2,3,\dots,n} d_j$  so that customer 1 always receives the delivery at the optimal solution. After the order for customer 1 is completed, it expires in  $B$  time units. Now, let  $\sum_{i=1,\dots,j-1} Y_{i,j} = X_j$ , where  $X_j = 1$  if customer  $j$  is selected to receive the delivery and  $X_j = 0$  otherwise,  $\forall j \in S \setminus \{1\}$ . Then,  $\mathbf{P}$  becomes:

$$\text{Max. } \sum_{j \in S \setminus \{1\}} d_j X_j,$$

*Subject to:*

$$\sum_{j \in S \setminus \{1\}} d_j X_j \leq r \cdot B, \quad X_j \in \{0,1\}, \quad \forall j \in S \setminus \{1\},$$

or a binary knapsack problem.

$\mathbf{P}$  has some interesting properties. To identify these properties, let  $\sigma = \langle [1], [2], \dots, [m] \rangle$ ,  $m \leq n$ ,  $1 \leq [i] \leq n$ ,  $1 \leq i \leq m$ , and  $\sigma \subset S$ , be a subset of  $S$ , which preserves the order of  $S$  and defines a production and transportation *schedule*. For any given schedule  $\sigma$  and customer  $[j]$ ,  $[j] \in \sigma$ , let's calculate the following:

$$\text{Earliest delivery time at } [j]: \quad T_{[j]} = \max\{a_{[j]}, T_{[j-1]} + \tau_{[j-1],[j]}\} \quad (7)$$

$$\text{Waiting time at } [j]: \quad w_{[j]} = \max\{0, a_{[j]} - T_{[j-1]} - \tau_{[j-1],[j]}\} \quad (8)$$

$$\text{Tardiness at } [j]: \quad \Delta_{[j]} = \max\{0, T_{[j]} - b_{[j]}\} \quad (9)$$

$$\text{Life-span lateness of } [j]: \quad l_{[j]} = \max\{0, T_{[j]} - [\sum_{\forall [i] \in \sigma, i \leq j} d_{[i]} / r + B]\} \quad (10)$$

where in (10), quantity  $\sum_{\forall [i] \in \sigma, i \leq j} d_{[i]} / r$  defines the time point at which the life

span of the order for customer  $[j]$  starts. Let  $T_{[0]} = T_0 = (\sum_{i \in \sigma} d_{[i]})/r$  stands for the time that the truck leaves the plant (i.e., node 0). If  $\sigma$  is a *feasible* schedule, then it must satisfy constraints (2)-(6), and ensure  $\Delta_i = 0, l_i = 0$ , for  $\forall i \in \sigma$ . Let

$$G(\sigma) = \sum_{i \in \sigma} d_i$$

be the objective value of a feasible  $\sigma$ . In addition, we assume that the triangle inequality holds for the traveling times, or  $\tau_{i,k} + \tau_{k,j} \geq \tau_{i,j}$ ,  $\forall i, j, k \in S \cup \{0\}$ .

**Definition 1.** (The *slack* of node  $j$ ) For any given schedule  $\sigma$  and node  $j$ ,  $j \in \sigma$ , the *slack* of node  $j$  is defined as  $s_j = T_j - a_j$ .

By definition 1, slack  $s_j$  denotes the maximum reduction in the delivery time,  $T_j$ , at node  $j$  (i.e., customer  $j$ ) achieved by deleting some other node(s) to make the truck deliver to  $j$  earlier, which however cannot be earlier than  $a_j$ . Now, let  $Z$  be an arbitrary tardy node in  $\sigma$  with  $\Delta_Z > 0$ , and node  $j$  be a predecessor of  $Z$  with a slack  $s_j \geq 0$  (see **Figure 2**), and consider the following result.

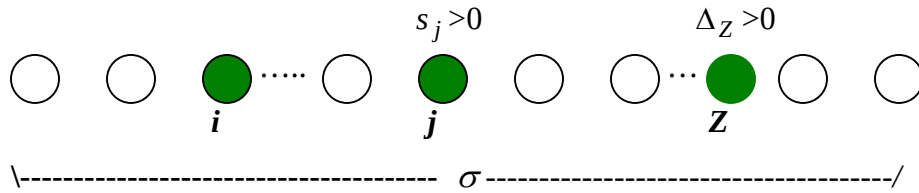


**Figure 2.** An illustrative example of **Proposition 1**.

**Proposition 1:** For any given schedule  $\sigma$  and any pair of nodes in  $\sigma$ ,  $j$  and  $Z$ , with  $j < Z$  and  $\Delta_Z > 0$ . If  $s_j < \Delta_Z$  then deleting node  $i$ , where  $i < j$  and  $i \in \sigma$ , does not change the infeasibility of  $Z$ .

**Proof.** Let  $T'_j$ , and  $\Delta'_Z$ , be the new delivery time to node  $j$ , and the new tardiness of node  $Z$ , after node  $i$  is deleted, Then, we have  $T'_j \leq T_j$  by triangle inequality, and  $T_Z - T'_Z \leq T_j - T'_j$  by (7) and the fact that  $j < Z$ . Since

$$\Delta'_Z = \Delta_Z - (T_Z - T'_Z) \geq \Delta_Z - (T_j - T'_j) \geq \Delta_Z - s_j$$

and  $\Delta_Z - s_j > 0$ , we have  $\Delta'_Z > 0$ , and the infeasibility of node  $Z$  remains after  $i$  is deleted.  $\diamond$

Proposition 1 indicates that it is inefficient to delete node  $i$  to exchange for the feasibility of node  $Z$  if there exists a node  $j$  between nodes  $i$  and  $Z$  that  $0 \leq s_j < \Delta_Z$ . The results of Proposition 1 also leads to the claim by Proposition 2 below (with proof skipped).

**Proposition 2.** *For any given schedule  $\sigma$  and node  $k \in \sigma$ , as other nodes are deleted from  $\sigma$ , the slack and the delivery time of node  $k$  monotonically decrease.*

**Example 2.** Consider the time windows and traveling time between the locations of four consecutive customers,  $i-1$ ,  $i$ ,  $i+1$  and  $i+2$  in a given schedule  $\sigma$  (see Figure 3). Suppose the delivery time to node  $i-1$  is  $T_{i-1} = a_{i-1} = 80$ . Then, before node  $i$  is deleted, we have

$$w_i = 8, \quad T_i = 98, \quad s_i = 0, \quad \Delta_i = 0$$

$$w_{i+1} = 0, \quad T_{i+1} = 113, \quad s_{i+1} = 13, \quad \Delta_{i+1} = 0$$

$$w_{i+2} = 0, \quad T_{i+2} = 125, \quad s_{i+2} = 9, \quad \Delta_{i+2} = 4$$

After node  $i$  is deleted, we have

$$w'_{i+1} = 2, \quad T'_{i+1} = 100, \quad w'_{i+2} = 4, \quad T'_{i+2} = 116, \quad \text{and} \quad s'_{i+1} = s'_{i+2} = 0.$$

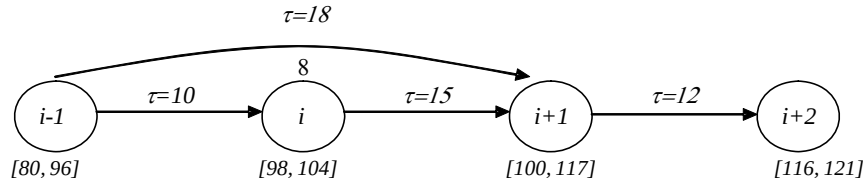


**Figure 3. An illustrative example of Proposition 2**

Now, let  $X = \langle X_1, X_2, \dots, X_n \rangle \in \{0, 1\}^n$  be a binary vector. Let  $\Delta(X) = \sum_{1 \leq i \leq n} \Delta_i \cdot X_i$ ,  $L(X) = \sum_{1 \leq i \leq n} l_i \cdot X_i$ , and  $G(X) = \sum_{1 \leq i \leq n} d_i \cdot X_i$ , where  $\Delta_i$ , and  $l_i$ , are defined by (9), and (10), respectively.

**Proposition 3 (Condition of optimality).** *If  $X^* = \langle X_1^*, X_2^*, \dots, X_n^* \rangle$  is an optimal solution to  $\mathbf{P}$ , then for every binary vector  $X = \langle X_1, X_2, \dots, X_n \rangle \in \{0, 1\}^n$ , we have either  $\Delta(X) > 0$ ,  $L(X) > 0$ , or  $G(X) \leq G(X^*)$ .*

An optimal solution to  $\mathbf{P}$ ,  $X^*$ , defines a subset of  $S$

$$\sigma^* = \langle [1], [2], \dots, [m] \rangle, \quad m \leq n, 1 \leq [i] \leq n, \forall [i] \in \sigma^*$$

so that  $[i] \in \sigma^*$  if and only if  $X_i^* = 1$ . The property of optimality stated in Proposition 3 implies that exchanging any element of  $\sigma^*$  for any element in  $S / \sigma^*$  either violates the feasibility or does not improve the value of  $G(\sigma^*)$ . This condition together with the properties of  $\mathbf{P}$  stated in Propositions 1 and 2 lay a foundation for our greedy search algorithm to be discussed in the next section.

### 3. A greedy Heuristic for $\mathbf{P}$

To start, let  $\sigma^*$  be the optimal schedule that solves  $\mathbf{P}$ ,  $\phi^* = S / \sigma^*$  be the subset of customers/nodes deleted under schedule  $\sigma^*$ , and  $G(\sigma^*)$  be the maximum amount of feasibly satisfied demand achieved by  $\sigma^*$ .

The heuristic that we propose for solving  $\mathbf{P}$  searches for  $\phi^*$ , instead of  $\sigma^*$ , via an iterative search process. Each iteration starts with a given schedule  $\sigma$  and initially  $\sigma = S$ . Let  $G(\sigma)$  be the total amount of *feasibly* satisfied demand by  $\sigma$ , then  $G(\sigma) = \sum_{j \in \sigma} d_j$  if schedule  $\sigma$  is feasible and  $G(\sigma) < \sum_{j \in \sigma} d_j$  if  $\sigma$  contains at least one infeasible delivery. Since some previously infeasible nodes may become feasible as we delete an infeasible node from  $\sigma$ , the delivery times to nodes during the calculation for the value of  $G(\sigma)$  are updated whenever a node is deleted (starting from the first infeasible node in a given  $\sigma$ ).

During each iteration, our proposed heuristic, called *Algorithm H*, searches for the pair of nodes,  $(i^*, k^*)$ , where  $1 \leq i^* \leq k^* \leq n$ ,  $(i^*, k^*) \subset \sigma$ , and  $i^* = k^*$  is allowed, to delete from  $\sigma$  so that the value of  $G$  can be improved by the most. That is

$$G(\sigma / \{i^*, k^*\}) = \text{Max.} \{G(\sigma / \{i, k\}) \mid \forall i, k \in \sigma\} \quad (11)$$

After nodes  $(i^*, k^*)$  are permanently deleted from  $\sigma$ , the reduced schedule  $\sigma' = \sigma / \{i^*, k^*\}$  becomes the given schedule for the next iteration. The search continues until  $\sigma$  becomes feasible and the deliveries to the remaining nodes in  $\sigma$  satisfy (2)-(6).

As we discussed in section 2, when  $n$  is large, an exhaustive enumeration of all possible pairs of  $(i, k)$  in each iteration for identifying  $(i^*, k^*)$  could be inefficient. To handle this issue, the following result together with Proposition 1 can be applied to speed up the search for  $(i^*, k^*)$  in each iteration without sacrificing the solution quality.

Let  $\sigma$  be any given candidate schedule containing nodes  $i', i, v$  and  $z$ , where  $i' < i < v < z$  and  $z$  is a tardy node/customer.

**Proposition 4.** *For any given candidate schedule  $\sigma$  and nodes  $i' < i < v < z$  in  $\sigma$  with  $s_v < \Delta_z$ , if  $|\phi^*| \leq 2$  and  $z \in \sigma^*$ , then  $\phi^* \neq \{i', i\}$ .*

**Proof.** (The proof is similar to that for Proposition 1 and thus skipped).

Proposition 4 implies the following. If  $|\phi^*| \leq 2$ ,  $z$  is a tardy node (see Figure 2 in section 2) and  $v$  is the nearest predecessor of  $z$  with  $s_v < \Delta_z$ , then the optimal solution  $(i^*, k^*)$  can only be one of the following pairs:

$$1. \{(i, k) | 1 \leq i \leq v, k = v\} \quad (12a)$$

$$2. \{(i, k) | 1 \leq i < v, v < k < z\} \quad (12b)$$

$$3. \{(i, k) | v \leq i \leq k \leq n\} \quad (12c)$$

Let  $\sigma$ ,  $|\sigma| \leq |S|$ , be the candidate schedule to be evaluated,  $\phi(\sigma)$  be the set of nodes that are deleted from  $\sigma$  during the search process by H, and  $G(\sigma)$  be the total amount of *feasibly* satisfied demand with  $G(\sigma) \leq \sum_{j \in \sigma} d_j$ . Then, the search algorithm H can be outlined as follows.

---

**Algorithm H:** {Initially, let  $\sigma = S$ ,  $\phi(\sigma) = \emptyset$ }

{Input:  $\sigma$ ,  $\phi(\sigma)$ , and  $G(\sigma)$ , and let  $G^{low} = G(\sigma)$ }

**Step 1.** Given current schedule  $\sigma$ , evaluate  $\phi(\sigma)$ . If  $\phi(\sigma) = \emptyset$ , terminate the search and output  $\sigma$  with  $G^H = G(\sigma)$ . Otherwise, for the first  $Z$ ,  $Z \in \phi(\sigma)$ , applying (12a)-(12c) to choose the pair of nodes,  $(i^*, k^*)$ , to permanently delete from the given  $\sigma$ .

*//Applying (12a)-(12c) to choose  $(i^*, k^*)$  excludes those pairs of nodes that are inferior//*

**Step 2.** If  $G(\sigma / \{i^*, k^*\}) > G^{low}$ , then permanently eliminate  $(i^*, k^*)$  from  $\sigma$ , and let

$\sigma \leftarrow \sigma / \{i^*, k^*\}$  and  $G^{low} = G(\sigma / \{i^*, k^*\})$ . If  $G(\sigma / \{i^*, k^*\}) = G^{low}$ , then eliminate tardy node  $Z$  from  $\sigma$ , and let  $\sigma \leftarrow \sigma / \{Z\}$  without changing the value of  $G^{low}$ . Return to Step 1;

---

Note that each iteration permanently deletes at least one node, the search by algorithm H terminates therefore in no more than  $n$  iterations. Since within each iteration, the number of pairs of nodes being evaluated is bounded from above by  $O(n^2)$ , the computational complexity of Algorithm H is thus  $O(n^3)$ . Also note that during the search process, the inequalities

$$G(S) \leq G(\sigma_1) \leq \dots \leq G(\sigma_i) \leq G(\sigma_{i+1}) \leq \dots \leq G^H$$

hold, where  $\sigma_i$  and  $\sigma_{i+1}$  are schedules to be evaluated in two consecutive iterations during the search process. In addition, we have the following result (proof skipped).

**Proposition 5.** (Conditional Optimality of Algorithm H) If  $\phi^* \leq 2$ , then  $\sigma_1 = \sigma^*$  and algorithm H finds the optimal solution to  $\mathbf{P}$  after the first iteration.

However, as we can see, if  $\phi^* > 2$ , then the optimality of the solution by Algorithm H is no longer guaranteed. In that case, the algorithm terminates with a greedy feasible solution. Example 3 below illustrates the use of Algorithm H to search for a feasible solution to  $\mathbf{P}$ .

**Example 3.** Let the given customer sequence be  $S = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$ ,  $B = 630$ , and  $r = 10$ , with following set of customer demand and time windows:

| Customer j | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $d_k$      | 90  | 29  | 53  | 27  | 36  | 89  | 45  | 18  | 84  | 38  |
| $a_k$      | 128 | 182 | 274 | 298 | 388 | 480 | 534 | 538 | 636 | 726 |
| $b_k$      | 144 | 239 | 288 | 385 | 447 | 487 | 560 | 603 | 657 | 756 |

Iteration 1: Given the initial schedule  $\sigma = S = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$ , we find  $i^* = 4, j^* = 8$  and  $G(\sigma / \{4, 8\}) = \max\{G(\sigma / \{i, j\}) \mid \forall i, j \in \sigma\} = 406$ . We permanently delete these two nodes, which gives  $\sigma = \langle 1, 2, 3, 5, 6, 7, 9, 10 \rangle$  for the next iteration.



Iteration 2: Given schedule  $\sigma = \langle 1, 2, 3, 5, 6, 7, 9, 10 \rangle$ , we find  $i^* = j^* = 10$  and  $G(\sigma / \{10\}) = \max\{G(\sigma / \{i, j\}) \mid \forall i, j \in \sigma\} = 406$  with  $\Delta_k = 0, l_k = 0, \forall k \in \sigma / \{10\}$ . Permanently delete node 10 from  $\sigma$  and stop the search with schedule  $\sigma = \langle 1, 2, 3, 5, 6, 7, 9 \rangle$ , which is also the optimal solution to **P** in this example.

To observe the empirical performance of Algorithm H, we randomly generated 490 test cases where the number of customers,  $n$ , ranged from 10 to 80. The customer demands were sampled from Uniform (100, 1500). The production rate,  $r$ , ranged from 10 to 50, while the product life span,  $B$ , ranged from 500 to 800. The truck traveling times were generated based upon Euclidean norm distances. Each customer time window was defined by a beginning time,  $a_i$ , and an ending time,  $b_i, 1 \leq i \leq n$ , where the values of  $a_i$  were randomly sampled from a Uniform distribution with a mean equal to the total truck travel time from plant to customer  $i$  along the given sequence of  $n$  customers,  $S$ . The widths of customer time windows were randomly sampled from Uniform (0.05 $T$ , 0.2 $T$ ), where parameter  $T$  stands for the total travel time needed to visit all the  $n$  customers. Given the randomly generated width for each customer window,  $\omega_i$ , the value of  $b_i, 1 \leq i \leq n$ , was then set to be  $a_i + \omega_i$ .

| n  | 5% Unfulfillment Rate |                |             | 15% Unfulfillment Rate |                |             | 20% Unfulfillment Rate |                |             |
|----|-----------------------|----------------|-------------|------------------------|----------------|-------------|------------------------|----------------|-------------|
|    | CPLEX time            | Heuristic time | Average Gap | CPLEX time             | Heuristic time | Average Gap | CPLEX time             | Heuristic time | Average Gap |
| 10 | 0.18                  | 1.42           | 0.00%       | 0.29                   | 1.99           | 0.29%       | 0.33                   | 2.87           | 0.68%       |
| 20 | 20.64                 | 3.33           | 0.00%       | 33.94                  | 6.69           | 1.06%       | 52.37                  | 9.98           | 1.86%       |
| 30 | 109.1                 | 5.15           | 0.17%       | 1200                   | 11.38          | 2.33%       | 2481                   | 20.89          | 2.50%       |
| 40 | 1506                  | 9.47           | 0.19%       | 3600                   | 19.77          | 3.16%       | 3600                   | 27.90          | 4.16%       |
| 50 | 3227                  | 13.40          | -0.17%      | 3600                   | 22.39          | 6.21%       | 3600                   | 36.18          | 9.33%       |
| 60 | 3600                  | 20.64          | -1.26%      | 3600                   | 38.99          | 8.07%       | 3600                   | 51.33          | 12.21%      |
| 70 | 3600                  | 39.66          | -2.03%      | 3600                   | 50.10          | 9.19%       | 3600                   | 60.05          | 11.58%      |
| 80 | 3600                  | 45.98          | -2.42%      | 3600                   | 55.11          | 9.78%       | 3600                   | 66.42          | 12.16%      |

**Table 3.** Empirical Observations on the Performance of Algorithm H.

Table 3 shows the average deviation of the heuristic solutions from the optimal solutions obtained by CPLEX Solver (on a 1.2G Mhz Dell Latitude). The empirical observations are listed against the number of customers and the *average un-fulfillment ratio* at the levels of 5%, 15% and 20%, respectively, where the average un-fulfillment ratio stands for  $|\varphi^*|/|S|$  averaged from ten test cases generated for each observation reported here. The required search time by CPLEX

Solver increases quickly as the number of customers goes beyond 40. Therefore, we set a one-hour CPU time limit so that whenever the solution time by CPLEX exceeds the limit, the best feasible solution obtained so far is used as a surrogate. Since algorithm  $H$  guarantees to find the optimal solution for the case with  $|\varphi^*| \leq 2$  and usually finds a close-to-optimal solution if the ratio of  $|\varphi^*|/|S|$  is low, the deviations from the optimal solution (i.e., the error gaps) with respect to the average ratio  $|\varphi^*|/|S| = 5\%$  were minimal. In a number of cases, we noticed that the solutions by algorithm  $H$  outperformed the surrogates of optimal solutions. The magnitude of error gaps, however, increases as the average unfulfillment ratio,  $|\varphi^*|/|S|$ , and the number of customers,  $n$ , increase. When this ratio increases to 20% and  $n$  goes beyond 60, the deviation from optimal solutions (or from the surrogate of optimal solutions) exceeds 12%. This is due to the myopic nature, as all the greedy heuristics, of algorithm  $H$  which focuses only on the nodes that make the most immediate improvement of the objective value. When the number of nodes that needs to be deleted at the optimal solution is large, the chance that Algorithm  $H$  fails to identify the right nodes increases. A graphical plot of the results reported in Table 3 is given in Figure 5 below.

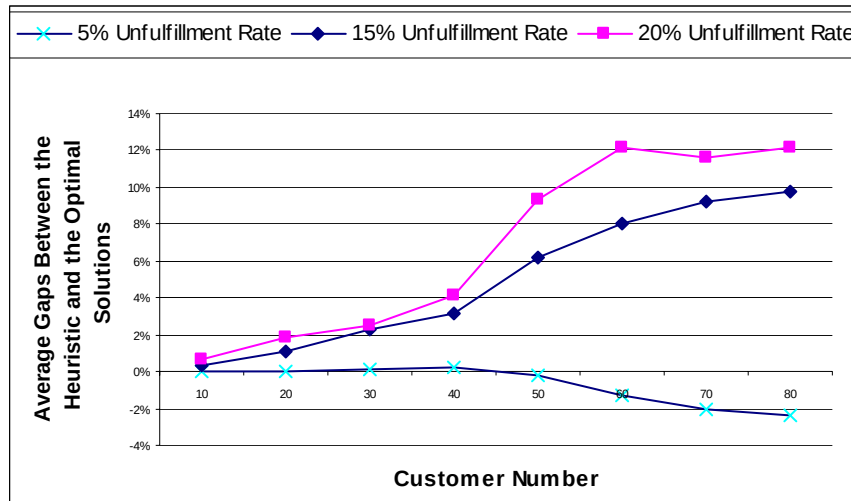


**Figure 5.** Empirical Observations on the Performance of Algorithm  $H$ .

Tables 4a and 4b present the deviations of the heuristic solution from the CPLEX solution, against relative standard deviations of the demand, and relative widths of customer receiving time windows, respectively. The results (Table 4a) show that the heterogeneity in customer demand has a negative impact on the

| Relative standard deviation of customer demand<br>$= \frac{\text{stdev}\{\text{demand}\}}{\text{average}\{\text{demand}\}}$ | n=30    |         |              | n=40    |         |              |
|-----------------------------------------------------------------------------------------------------------------------------|---------|---------|--------------|---------|---------|--------------|
|                                                                                                                             | min gap | max gap | average gaps | min gap | max gap | average gaps |
| 10%                                                                                                                         | 0.00%   | 4.41%   | 1.82%        | 0.00%   | 6.38%   | 2.06%        |
| 15%                                                                                                                         | 0.00%   | 4.74%   | 2.07%        | 0.00%   | 5.24%   | 2.72%        |
| 20%                                                                                                                         | 0.04%   | 4.66%   | 2.50%        | 1.17%   | 7.96%   | 4.16%        |
| 25%                                                                                                                         | 1.50%   | 5.12%   | 3.61%        | 2.62%   | 7.34%   | 5.79%        |
| 30%                                                                                                                         | 3.20%   | 7.76%   | 5.16%        | 5.75%   | 10.11%  | 8.10%        |

**Table 4a.** Average error gaps against relative standard deviation of demand.

| Relative mean width of time windows<br>$= \frac{\text{mean width of time windows}}{\text{total travel time}}$ | n=30    |         |              | n=40    |         |              |
|---------------------------------------------------------------------------------------------------------------|---------|---------|--------------|---------|---------|--------------|
|                                                                                                               | min gap | max gap | average gaps | min gap | max gap | average gaps |
| 5%                                                                                                            | 4.12%   | 7.95%   | 5.24%        | 4.17%   | 8.64%   | 5.89%        |
| 15%                                                                                                           | 1.31%   | 5.08%   | 3.26%        | 3.52%   | 7.95%   | 4.92%        |
| 25%                                                                                                           | 0.56%   | 3.19%   | 2.50%        | 2.16%   | 4.77%   | 4.16%        |
| 35%                                                                                                           | 0.00%   | 2.84%   | 1.43%        | 0.88%   | 4.94%   | 3.17%        |
| 45%                                                                                                           | 0.00%   | 1.50%   | 0.92%        | 0.00%   | 3.20%   | 2.09%        |

**Table 4b.** Average error gaps against relative mean width of time windows.

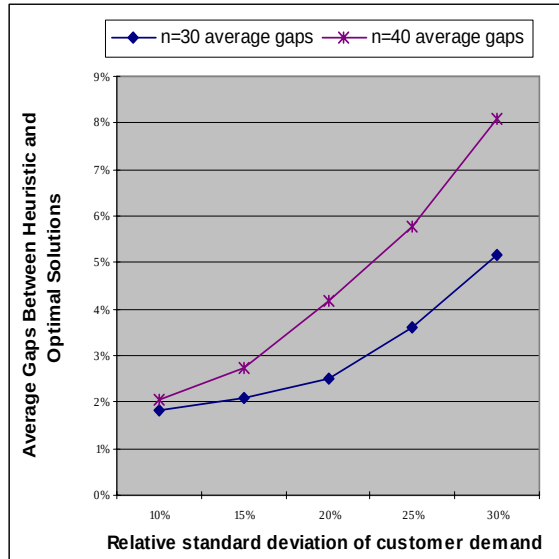


**Figure 6a.** Performance against relative Standard deviations of demand.

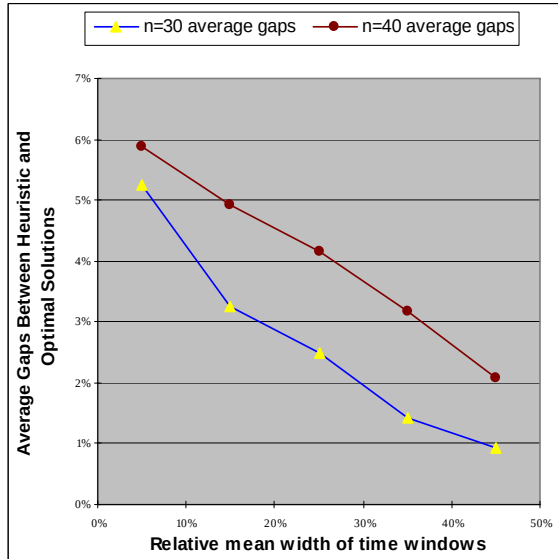


**Figure 6b.** Performance against Relative window widths.

performance of algorithm  $H$ . This is because the heterogeneity of customer demands, so the time to produce customer orders, together with the heterogeneity of customer time windows and truck traveling times between customers make it difficult for the greedy heuristic to identify the right subset of customers to delete

from the schedule. On the other hand, the results in Table 4b show that as the width of customer receiving time windows becomes more relaxed (i.e., increases), the performance of the heuristic improves. As the ratio of mean window width to the total travel time increases to the level of 45%, the average deviation from the optimal CPLEX solution reduces to be within 2.1% of the optimal CPLEX solution, even with  $n=40$  customers and a relatively high level of average un-fulfillment rate  $|\varphi^*|/|S|=20\%$ . This implies that, in reality, negotiating with customers to increase the width of their receiving time windows has potential to improve the operation performance.

#### 4. Concluding remarks

We studied the single-truck production and distribution problem with a short shelf-life product, a given sequence of customers, a limited production capacity, and delivery time window constraints. The problem is to choose a subset of customers to receive the deliveries so that the total amount of satisfied demand is maximized while all the constraints are satisfied. We showed that the general version of this problem is NP-hard even with relaxed time windows and instantaneous truck traveling time, and analyzed the problem properties and the conditions under which the optimality of a solution is achieved. A greedy heuristic is proposed based on this analysis. Empirical observations on the error gaps achieved by this heuristic are reported.

The work we presented in this paper can be extended in several directions. When the delivery truck has a limited capacity, there is a customer partition issue by which the given customer sequence must be partitioned into several disjoint trips so that the total demand handled by each trip does not exceed the truck capacity. The problem is then how to partition to maximize the total amount of satisfied demand. The problem **P** considered in this paper is a sub-problem of the zero-inventory PDP. A more general problem would allow the truck route to be optimized. Furthermore, we have only considered maximizing the total demand to be satisfied. There are many other objectives of interests from both a theoretical and a practical point of view. Examples of such objectives could include minimizing the fleet size, the deviation of actual delivery time from the customer receiving time windows, and the maximum time interval between the production completion time and the delivery time at a customer site subject to a fixed fleet size.

## References

1. Ghiani, G, G. Laporte and R. Musmanno (2004) *Introduction to Logistics Systems Planning and Control*. Wiley Publishing Company.
2. Potts, C. N. (1980) An Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Time. *Operations Research*. **28**, 1436-1441.
3. Hall, L and D. Shmoys (1989) Approximation Schemes for Constrained Scheduling Problems. *Proceedings of the 30<sup>th</sup> Annual Symposium on Foundations of Computer Science*. 134-140.
4. Woeginger, G. (1998) A Polynomial-Time Approximation Scheme for Single-Machine Sequencing with Delivery Times and Sequence-Independent Batch Set-up Times. *Journal of Scheduling*. **1**, 79-87.
5. Zdrzalka, S. (1995) Analysis of approximation algorithms for single-machine scheduling with delivery times and sequence independent batch setup times. *European Journal of Operational Research*. **80**, 371-380.
6. Lee, C-Y. and Z-L. Chen (2001) Machine Scheduling with Transportation Considerations. *Journal of Scheduling*. **4**, 3-24.
7. Pinedo, M. (1995) *Scheduling: Theory, Algorithms, and Systems*.. Prentice Hall.
8. Cheng, T.C. and V. Gordon (1994) Batch Delivery Scheduling on a Single Machine. *Journal of the Operational Research Society*. **45**, 1211-1216.
9. Cheng, T. C., V. Gordon and M. Kovalyov (1996) Single Machine Scheduling with Batch Deliveries. *European Journal of Operations Research*. **94**, 277-283.
10. Cheng, T.C., M. Kovalyov and B. Lin (1994) Single Machine Scheduling to Minimize Batch Delivery and Job Earliness Penalties. *SIAM J. Optimization*. **7**, 547-559.
11. Herrmann, J.W. and C-Y. Lee (1993) On scheduling to minimize earliness-tardiness and batch delivery costs with a common due date. *European Journal of Operational Research*. **70**, 272-288.
12. Yuan, J. A. (1996) Note on the Complexity of Single-Machine Scheduling with a Common Due Date, Earliness-Tardiness, and Batch Delivery Costs. *European Journal of Operational Research*. **94**, 203-205.
13. Wang, G. and T.C. Cheng (2000) Parallel Machine Scheduling with Batch Delivery Costs. *International Journal of Production and Economics*. **68**, 177-183.
14. Hall, N.G. and C. N. Potts (2004) Supply Chain Scheduling: Batching and Delivery. *Operations Research*. (to appear).
15. Chen, Z-L. (2002) *Integrated Production and Distribution Operations: Taxonomy, Models, and Review*. *Supply Chain Analysis in the E-Business Era*. Kluwer Academic Publishers.
16. Hurter, A.P. and M. G. Van Buer (1996) The Newspaper Production/Distribution Problem. *Journal of Business Logistics*. **17**, 85-107.
17. Van Buer, M.G., D.L. Woodruff and R.T. Olson (1999) Solving the Medium Newspaper Production/Distribution Problem. *European Journal of Operations Research*. **115**, 237-253.
18. Geismar, N, L. Lei and C. Sriskandarajah (2003) The Integrated Production and Transportation Scheduling Problem with Expiring Product and Non-Instantaneous Transportation Time. *Transportation Science*. (Under the review).
19. Chen, Z-L. (1996) Scheduling and Common Due Date Assignment with Earliness-Tardiness Penalties and Batch Delivery Costs. *European Journal of Operations Research*. **93**, 49-60.

---

# Scheduling UET–UCT task systems under the out-forest precedence constraints

Claire Hanen<sup>1</sup> and Yakov Zinder<sup>2</sup>

<sup>1</sup> LIP6, Université Pierre et Marie Curie, Paris, France, [han@u-paris10.fr](mailto:han@u-paris10.fr)

<sup>2</sup> Department of Mathematics, Faculty of Science, University of Technology,  
Sydney, Australia, [yakov.zinder@uts.edu.au](mailto:yakov.zinder@uts.edu.au)

**Summary.** This paper is concerned with two problems of scheduling unit execution time task systems on parallel identical processors with unit communication delays and precedence constraints in the form of an out-forest. One problem is the maximum lateness problem with different release times. To the authors' knowledge, the presented worst-case analysis of an approximation algorithm, generalizing the idea of the classical Garey-Johnson algorithm, is the first result of this type for models with communication delays and different release times, and provides the best currently known performance guarantee for problems with communication delays and precedence constraints in the form of an out-forest. The paper also presents an iterative algorithm, constructing an optimal schedule, for a two-processor system, for the criterion of total completion time, among optimal schedules for the criterion of maximum lateness.

**Key words:** machine scheduling, parallel processors, communication delay, precedence constraints, worst-case analysis, maximum lateness

## 1 Models with UET–UCT task systems

The scheduling model with parallel identical processors, precedence constraints, unit execution time (UET) tasks and unit communication delays is one of the key models in scheduling theory. Being important from the practical viewpoint in its own right, this model also serves as a testing ground for new ideas, methods, and scheduling techniques. The model can be described as follows. A set  $N = \{1, \dots, n\}$  of  $n$  tasks is to be processed on  $m > 1$  parallel identical processors (machines). Each processor can process at most one task at a time, and each task can be processed by any processor. Once a processor begins executing a task, it continues until the completion, i.e. no preemptions are allowed. Each task requires one unit of processor's time, and the processing of tasks commences at time  $t = 0$ . The restrictions on the order in which tasks can be processed are specified in the form of precedence constraints: an anti-reflexive, anti-symmetric and transitive relation on  $N$ . If task  $i$  precedes task  $j$ , denoted  $i \rightarrow j$ , then the processing of task  $i$  must be completed before the processing of task  $j$  begins. Moreover, if these two tasks are processed on different

processors, then the processing of  $j$  can commence not earlier than one time unit after the completion of  $i$ . This delay is referred to as an unit communication time (UCT) or unit communication delay.

Schedule  $\sigma$  is a mapping  $N \Rightarrow Z_+$ , where  $Z_+$  is the set of all positive integers, which specifies for each task  $j \in N$  its completion time  $C_j(\sigma)$ . The existence of communication delays implies that in any feasible schedule  $\sigma$ ,

- for any three tasks  $i, j$  and  $g$  such that  $i \rightarrow j$  and  $i \rightarrow g$ ,

$$C_i(\sigma) \leq \max[C_j(\sigma), C_g(\sigma)] - 2;$$

- for any three tasks  $i, j$  and  $g$  such that  $i \rightarrow g$  and  $j \rightarrow g$ ,

$$\min[C_i(\sigma), C_j(\sigma)] \leq C_g(\sigma) - 2.$$

The goal is to find a schedule minimizing some criterion  $\gamma(C_1(\sigma), \dots, C_n(\sigma))$ . In the standard three-field notation (see for example [Br01]) this problem is denoted by  $P|prec, p_j = 1, c_{ij} = 1|\gamma$ , where the term  $c_{ij} = 1$  indicates the existence of unit communication delays if any two tasks  $i$  and  $j$  are processed on different processors.

One of the most commonly used criteria is the criterion of total completion time

$$C_\Sigma(\sigma) = \sum_{j \in N} C_j(\sigma).$$

Another frequently used criterion is the criterion of maximum lateness

$$L_{\max}(\sigma) = \max_{j \in N} [C_j(\sigma) - d_j],$$

where  $d_j$  is an integer due date associated with task  $j$ . If all due dates are equal to zero, then the criterion of maximum lateness converts to the makespan

$$C_{\max}(\sigma) = \max_{j \in N} C_j(\sigma).$$

Another maximum lateness related criterion is the criterion of maximum tardiness

$$T_{\max}(\sigma) = \max[0, L_{\max}(\sigma)].$$

A straightforward generalization of the  $P|prec, p_j = 1, c_{ij} = 1|\gamma$  problem is the problem with release times, which differs from  $P|prec, p_j = 1, c_{ij} = 1|\gamma$  only by the assumption that the processing of any task  $j$  can commence only after a given nonnegative integer release time  $r_j$ . In the three-field notation the problem with release times is denoted by  $P|prec, r_j, p_j = 1, c_{ij} = 1|\gamma$ .

Another generalization of the basic  $P|prec, p_j = 1, c_{ij} = 1|\gamma$  problem is a problem with two criteria,  $\gamma$  and  $\delta$ . In this case, the goal is a schedule optimal for  $\delta$  among all schedules optimal for  $\gamma$ . The corresponding scheduling problem will be denoted by  $P|prec, p_j = 1, c_{ij} = 1|<\gamma, \delta>$ .

If the assumption of communication delays is waived, then the  $P|prec, p_j = 1, c_{ij} = 1|\gamma$  and  $P|prec, r_j, p_j = 1, c_{ij} = 1|\gamma$  problems become the classical parallel machine scheduling problems with precedence constraints and UET tasks, denoted  $P|prec, p_j = 1|\gamma$  and  $P|prec, r_j, p_j = 1|\gamma$ , respectively. In this case, the  $P|prec, p_j = 1, c_{ij} = 1|<\gamma, \delta>$  problem becomes  $P|prec, p_j = 1|<\gamma, \delta>$ .

## 2 The background and motivation

The  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$  problem is NP-hard [RSm87]. Moreover, in contrast to the  $P|prec, p_j = 1|C_{max}$  problem, which becomes trivial if there are unlimited (or as large as needed) number of processors, its counterpart with communication delays,  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$ , is NP-hard even if there are infinitely many processors [Pic95]. The  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$  problem with infinitely many processors is usually denoted by  $P\infty|prec, p_j = 1, c_{ij} = 1|C_{max}$ . The NP-hardness results for the models with communication delays motivated research on the worst-case performance of various approximation algorithms and studies of particular cases, defined either by a special structure of precedence constraints or by a restriction on the number of processors or by both.

The particular cases, defined by a restriction on the number of processors, are predominantly two-processor systems [FLMB96, LVV96, Sin04, Ver95, Ver00]. If there are only two processors, then in the three-field notation the term  $P$  in the first field is replaced by  $P2$ . Although a number of polynomial-time algorithms have been developed for the  $P2|prec, p_j = 1|C_{max}$ ,  $P2|prec, p_j = 1|L_{max}$  and  $P2|prec, r_j, p_j = 1|L_{max}$  problems [CG72, GJ76, GJ77, ZR98, Zin03], even the complexity status of  $P2|prec, p_j = 1, c_{ij} = 1|C_{max}$  remains yet unknown. On the other hand, most of the algorithms developed for the  $P2|prec, p_j = 1|C_{max}$ ,  $P2|prec, p_j = 1|L_{max}$  and  $P2|prec, r_j, p_j = 1|L_{max}$  problems allow modification for the models with communication delays and precedence constraints restricted to out-forests, i.e. partially ordered sets, where if  $i \rightarrow g$  and  $j \rightarrow g$ , then either  $i \rightarrow j$  or  $j \rightarrow i$ . If the precedence constraints are restricted to out-forests, then in the three-field notation the term  $prec$  in the second field is replaced by *out-forest*. Thus,

- as it has been shown in [Sin04], a modification of the algorithm, presented in [ZR98] for the  $P2|prec, p_j = 1|L_{max}$  problem, solves  $P2|out-forest, p_j = 1, c_{ij} = 1|L_{max}$ ;
- as it has been shown in [Ver00], a modification of the algorithm, presented in [GJ76] for the  $P2|prec, p_j = 1|L_{max}$  problem, solves  $P2|out-forest, p_j = 1, c_{ij} = 1|L_{max}$ ;
- as it has been shown in [Ver95], a modification of the algorithm, presented in [GJ77] for the  $P2|prec, r_j, p_j = 1|L_{max}$  problem, solves the corresponding problem with an out-forest and communication delays, i.e.  $P2|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$ .

The next logical step is an extension of the results for the two-processor models to the problems with two criteria. A polynomial-time algorithm for the  $P2|out-forest, p_j = 1, c_{ij} = 1|<L_{max}, C_\Sigma>$  problem is presented in Section 4. Relevant results for the models without communication delays can be found in [CG72], [Zin86], [LLRKS93], and [ZD03]. The algorithm, presented in [CG72], constructs a schedule optimal for  $P2|prec, p_j = 1|C_{max}$  and  $P2|prec, p_j = 1|C_\Sigma$  simultaneously [LLRKS93]. The algorithm, presented in [Zin86], solves  $P2|prec, r_j, p_j = 1|<L_{max}, C_\Sigma>$ . The algorithm, presented in [ZD03], constructs a schedule optimal for  $P2|prec, r_j, p_j = 1|C_{max}$  and  $P2|prec, r_j, p_j = 1|C_\Sigma$  simultaneously.

As it has been shown in [RSm87], for an arbitrary list schedule  $\sigma^L$ ,

$$C_{max}(\sigma^L) \leq \left(3 - \frac{2}{m}\right) C_{max}(\sigma^*) - \left(1 - \frac{1}{m}\right),$$



where  $\sigma^*$  is an optimal schedule for the  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$  problem. The best currently known performance guarantee for the  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$  problem was presented in [HM01], which describes an approximation algorithm, referred to in [HM01] as *FS* algorithm, and shows that

$$\frac{C_{max}(\sigma^{FS})}{C_{max}(\sigma^*)} \leq \frac{7}{3} - \frac{4}{3m},$$

where  $\sigma^{FS}$  is a schedule, constructed by the *FS* algorithm, and  $\sigma^*$  is an optimal schedule for the  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$  problem. This result in turn uses the result from [MK97], which studies the worst-case performance of an approximation algorithm for  $P\infty|prec, p_j = 1, c_{ij} = 1|C_{max}$ , based on the linear programming relaxation of the corresponding integer programming formulation.

Scheduling UET-UCT task systems, when the number of processors is a part of the problem input, is difficult even for restricted classes of precedence constraints. For example, as it has been shown in [LVV96], the  $P|prec, p_j = 1, c_{ij} = 1|C_{max}$  problem remains NP-hard even if the precedence constraints are restricted to in-forests, i.e. to the partially ordered sets where if  $i \rightarrow j$  and  $i \rightarrow g$ , then either  $j \rightarrow g$  or  $g \rightarrow j$ . Another example is  $P|out-forest, p_j = 1, c_{ij} = 1|L_{max}$ , which is also NP-hard.

The algorithm in [GJ76], which was originally developed for  $P2|prec, p_j = 1|L_{max}$  and which is commonly known as the Garey-Johnson algorithm for the model without release times, can be modified for the  $P|prec, p_j = 1|L_{max}$  problem. This modification in turn can be generalized for  $P|out-forest, p_j = 1, c_{ij} = 1|T_{max}$ , and according to [Ver00], satisfies the following performance guarantee

$$T_{max}(\sigma^{GJ}) \leq \left(2 - \frac{2}{m}\right) T_{max}(\sigma^*) + \left(1 - \frac{2}{m}\right) \max_{i \in N} d_i + 1 - \frac{2}{m}, \quad (1)$$

where  $\sigma^{GJ}$  is a schedule, constructed by the modified Garey-Johnson algorithm for the model without release times, and  $\sigma^*$  is an optimal schedule for the problem  $P|out-forest, p_j = 1, c_{ij} = 1|T_{max}$ . On the other hand, the algorithm in [GJ77], commonly known as the Garey-Johnson algorithm for the model with release times and originally developed for the  $P2|prec, r_j, p_j = 1|L_{max}$  problem, is more powerful than that in [GJ76] and can also be generalized for the problem  $P|prec, r_j, p_j = 1|L_{max}$  [HZ]. Therefore, its further generalization for the  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$  problem (and hence  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|T_{max}$ ) and an analysis of the worst-case performance of the resultant algorithm is a logical direction of further research.

### 3 A list algorithm and a list schedule

Analogously to the majority of scheduling algorithms for UET task systems, it is convenient to describe the algorithms, presented in Sections 4 and 5, using the notion of list algorithm. The problem  $P2|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$ , considered in Section 5, assumes different release times, whereas the problem  $P2|out-forest, p_j = 1, c_{ij} = 1|L_{max}$ , considered in Section 4, does not. The description below is given for the model with different release times and arbitrary number of processors.

The assumption that  $r_j = 0$ , for all  $j \in N$ , and that  $m = 2$  converts this description into one needed for Section 4. In what follows, for an arbitrary integer  $t$ , the time interval  $[t - 1, t]$  will be also referred to as a time slot  $t$ .

The list algorithm assumes that the tasks are arranged in a list. In constructing the corresponding schedule  $\sigma^L$ , referred to as a list schedule, the list algorithm iteratively selects a time slot, chooses tasks for processing in the current time slot, and eliminates them from the list. In the description of this algorithm,  $\mathcal{L}$  will denote the current list of tasks. If  $\mathcal{L} = (j_1, \dots, j_u)$ , then  $\mathcal{L}(j_k) = k$ . In other words, if  $j$  is in  $\mathcal{L}$ , then  $\mathcal{L}(j)$  gives its position in this list. Let  $M$  be an arbitrary subset of  $N$  and  $t$  be an arbitrary positive integer. Task  $j$  will be called available in time slot  $t$  with respect to  $M$ , if  $r_j < t$ ;  $C_i(\sigma^L) < t$  for all  $i$  such that  $i \rightarrow j$ ; and there exists no  $i \in M$  such that  $i \rightarrow j$ .

#### List Algorithm

1. Set  $t = \min_{j \in \mathcal{L}} r_j + 1$ ,  $M = \emptyset$ ,  $\kappa = 0$  and  $\pi = 1$ .
2. Starting with the  $\pi$ th element of the list, scan  $\mathcal{L}$  from left to right and find the first task  $j$  available in time slot  $t$  with respect to  $M$ . If  $j$  does not exist, then go to step 5.
3. Set  $C_j(\sigma^L) = t$ ,  $\pi = \mathcal{L}(j)$  and  $\kappa = \kappa + 1$ . If there exists task  $i$  such that  $C_i(\sigma^L) = t - 1$  and  $i \rightarrow j$ , then set  $M = M \cup \{i\}$ . Eliminate  $j$  from  $\mathcal{L}$ .
4. If  $\pi$  is less than or equal to the number of elements in  $\mathcal{L}$  and if  $\kappa < m$ , then go to step 2.
5. If  $\mathcal{L}$  is empty, then halt. Otherwise set  $t = \max[t + 1, \min_{j \in \mathcal{L}} r_j + 1]$ ,  $M = \emptyset$ ,  $\kappa = 0$  and  $\pi = 1$ , and go to step 2.

## 4 Scheduling with two criteria, $L_{max}$ and $C_\Sigma$

An algorithm for the  $P2|out-forest, p_j = 1, c_{ij} = 1|<L_{max}, C_\Sigma>$  problem, presented in this section, is based on [Ver00]. As it has been shown in [Ver00],  $P2|out-forest, p_j = 1, c_{ij} = 1|L_{max}$  can be solved using a modification of the Garey-Johnson algorithm for the model without release times [GJ76]. Using the original due dates  $d_1, \dots, d_n$ , the algorithm in [Ver00] calculates new due dates  $D_1, \dots, D_n$  with some special properties, including the following:

- $D_j \leq d_j$  for all  $j \in N$ ;
- for any schedule  $\sigma$

$$\max_{j \in N} [C_j(\sigma) - d_j] = \max_{j \in N} [C_j(\sigma) - D_j].$$

Then the algorithm, presented in [Ver00], constructs the resultant schedule  $\bar{\sigma}$ , using the List Algorithm and a list where the tasks are arranged in a nondecreasing order of their due dates  $D_j$ . Hence in [Ver00], the due dates  $D_1, \dots, D_n$  act as priorities for respective tasks, where a task with a smaller due date has a higher priority. The following algorithm refines this idea by giving a higher priority to a task with a smaller due date, and for any two tasks with the same due dates, to a task with a larger number of successors in the precedence constraints. For any  $i \in N$ , let  $K(i)$  be the set of all tasks  $j$  such that  $i \rightarrow j$ .

*Algorithm for  $P2|out-forest, p_j = 1, c_{ij} = 1|<L_{max}, C_\Sigma>$*

1. Set  $\tau = 0$ .
2. Using the due dates  $d_1, \dots, d_n$  and the algorithm presented in [Ver00], calculate due dates  $D_1, \dots, D_n$ . Arrange the tasks in a list  $\mathcal{L}$  such that, for any two tasks  $i$  and  $j$ , if  $\mathcal{L}(i) < \mathcal{L}(j)$ , then either  $D_i < D_j$ , or  $D_i = D_j$  and  $|K(i)| \geq |K(j)|$ . Construct the corresponding list schedule  $\bar{\sigma}$ .
3. If both processors are busy in every time slot  $t$ , satisfying the inequalities  $\tau + 1 < t < \max_{j \in N} C_j(\bar{\sigma})$ , then halt.
4. Among all integer  $t$ , satisfying the inequalities  $\tau + 1 < t < \max_{j \in N} C_j(\bar{\sigma})$  and such that in the time slot  $t$  only one processor is busy, select the smallest one and denote it by  $T$ .
5. Among all tasks  $i$  such that  $i \rightarrow j$  for all  $j$  satisfying the inequality  $C_j(\bar{\sigma}) > T$  select the task with the largest completion time. Denote this task by  $g$ . If

$$C_g(\bar{\sigma}) - D_g < \max_{j \in N} [C_j(\bar{\sigma}) - D_j],$$

then set  $d_j = D_j$ , for all  $j \in (N - \{g\})$ ,  $d_g = C_g(\bar{\sigma}) - \max_{j \in N} [C_j(\bar{\sigma}) - D_j]$ , and go to Step 2.

6. Set  $\tau = C_g(\bar{\sigma})$  and go to Step 3.

## 5 An algorithm for $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$ and its worst-case performance

This section is concerned with an algorithm for the  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$  problem, which bases on the idea of the Garey-Johnson algorithm for  $P2|prec, r_j, p_j = 1|L_{max}$  [GJ77]. The key concept of the Garey-Johnson algorithm is the notion of consistent due dates. In the presence of communication delays this notion also requires a modification (see also [Ver95]). In order to introduce a notion of consistent due dates, observe that if  $i \rightarrow j$  and  $r_i \geq r_j$ , then the replacement of  $r_j$  by  $r_i + 1$  does not affect any feasible schedule. Since for an out-forest all release times can be recalculated in  $O(n)$  operations, without loss of generality we will assume that  $i \rightarrow j$  implies the inequality  $r_i + 1 \leq r_j$ .

For any  $i \in N$  and any integers  $s$  and  $d$  such that

$$r_i \leq s \leq D_i \leq d, \tag{2}$$

we define two sets  $S(i, s, d)$  and  $R(i, s)$  as follows:

- $S(i, s, d)$  is the set of all tasks  $j$  such that  $j \neq i$ , and either  $i \rightarrow j$  or  $r_j \geq s$ ;
- $R(i, s)$  is the set of all tasks  $j \in N$  such that  $r_j = s$  and  $i \nrightarrow j$ , i.e.  $i$  does not precede  $j$  in the precedence constraints.

The due dates  $D_1, \dots, D_n$  are consistent if  $r_j \leq D_j - 1$  for all  $1 \leq j \leq n$  and for any  $i \in N$  and any integers  $s$  and  $d$ , satisfying (2),

1.  $|S(i, s, d)| \leq m(d - s)$ ;
2. if  $\left\lceil \frac{|S(i, s, d)|}{m} \right\rceil = d - s$ , then  $|S(i, s, d)| - |R(i, s)| \leq m(d - s - 1) + 1$  and  $D_i = s$ .

An arbitrary integer  $\Delta$  will be called feasible if there exists a set of integers  $D_1, \dots, D_n$  such that  $D_1, \dots, D_n$  are consistent and  $D_i \leq d_i + \Delta$  for all  $i \in N$ . If  $\Delta$  is feasible, then the corresponding consistent due dates  $D_1, \dots, D_n$  will be called  $\Delta$ -modified due dates. Let  $\Delta^*$  be the smallest among all feasible  $\Delta$ .

The following two lemmas demonstrate the significance of consistent due dates and  $\Delta^*$ -modified due dates in particular.

**Lemma 1.** *Let  $\sigma$  be an arbitrary schedule and  $D_j = C_j(\sigma)$ , for all  $j \in N$ , then  $D_1, \dots, D_n$  are consistent.*

**Lemma 2.** *Let  $\sigma^*$  be an optimal schedule for the  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$  problem, then*

$$\Delta^* \leq L_{max}(\sigma^*).$$

Because the scheduling algorithm for the  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$  problem which bases on the idea of the algorithm presented in [GJ77] for the  $P2|prec, r_j, p_j = 1|L_{max}$  problem, this algorithm will be referred to as Modified Garey-Johnson Algorithm.

#### Modified Garey-Johnson Algorithm

1. Compute  $\Delta^*$  and  $\Delta^*$ -modified due dates (using the approach described in [GJ77], this can be accomplished in  $O(n^3 \log_2 n)$  operations).
2. Construct a list schedule  $\bar{\sigma}$  for a list where the tasks are arranged in the non-decreasing order of there  $\Delta^*$ -modified due dates.

**Theorem 1.** *Let  $\sigma^*$  be an optimal schedule for the  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$  problem. Then*

$$L_{max}(\bar{\sigma}) \leq \left(2 - \frac{2}{m}\right) L_{max}(\sigma^*) + \left(1 - \frac{2}{m}\right) \max_{j \in N} d_j - r(m), \quad (3)$$

where

$$r(m) = \begin{cases} \frac{m-3}{m} & \text{for } m \text{ odd} \\ \frac{m-2}{m} & \text{for } m \text{ even.} \end{cases}$$

Besides establishing a performance guarantee for the  $P|out-forest, r_j, p_j = 1, c_{ij} = 1|L_{max}$  problem, Theorem 1 also improves the best previously known performance guarantee (1) for the problem of scheduling UET-UCT out-forests without release times and with the criterion of maximum tardiness. There exists an example, demonstrating that the upper bound (3) is asymptotically tight.

## References

- [Br01] Brucker, P.: Scheduling algorithms, third edition. Springer, Berlin (2001)
- [CG72] Coffman, E.G., Graham, R.L.: Optimal scheduling for two-processor systems. Acta Inform. **1**, 200–213 (1972)

- [FLMB96] Finta, L., Liu, Z., Milis, I., Bampis, E.: Scheduling UET-UCT series-parallel graphs on two processors. *Theoretical Computer Science*, **162**, 323–340 (1996)
- [GJ76] Garey, M. R., Johnson, D. S.: Scheduling tasks with nonuniform deadlines on two processors. *Journal of the Association for Computing Machinery*, **23**, 461–467 (1976)
- [GJ77] Garey, M. R., Johnson, D. S.: Two-processor scheduling with start-time and deadlines. *SIAM Journal on Computing*, **6**, 416–426 (1977)
- [HM01] Hanen, C., Munier, A.: An approximation algorithm for scheduling dependent tasks on  $m$  processors with small communication delays. *Discrete Applied Mathematics*, **108**, 239–257 (2001)
- [HZ] Hanen, C., Zinder, Y.: The worst-case analysis of the Garey-Johnson algorithm. submitted for publication.
- [LLRKS93] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and Scheduling: Algorithms and Complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (ed) *Logistics of Production and Inventory*, Elsevier Science Publishers B.V. (1993)
- [LVV96] Lenstra, J.K., Veldhorst, M., Veltman, B.: The complexity of scheduling trees with communication delays. *Journal of Algorithms*, **20**, 157–173 (1996)
- [MK97] Munier, A., König, J.-C.: A heuristic for scheduling problem with communication delays. *Operations Research*, **45**, 145–148 (1997)
- [Pic95] Picouleau, C.: Two new NP-complete scheduling problems with communication delays and unlimited number of processors. *Discrete Applied Mathematics*, **60**, 331–342 (1995)
- [RSm87] Rayward-Smith, V.J.: UET scheduling with interprocessor communication delays. *Discrete Applied Mathematics*, **18**, 55–71 (1987)
- [Sin04] Singh, G.: Scheduling UET-UCT outforests to minimize maximum lateness. *European Journal of Operational Research*, (2004)
- [Ver95] Verriet, J.: Scheduling UET, UCT dags with release dates and deadlines. Technical Report UU-CS-1995-31, Utrecht University (1995)
- [Ver00] Verriet, J.: Scheduling tree-like task systems with non-uniform deadlines subject to unit-length communication delays. *Discrete Applied Mathematics*, **101**, 269–289 (2000)
- [Zin03] Zinder, Y.: An iterative algorithm for scheduling UET tasks with due dates and release times. *European Journal of Operational Research* **149**, 404–416 (2003)
- [Zin86] Zinder, Y.: An efficient algorithm for deterministic scheduling problem with parallel machines. *Cybernetics*, N2, (1986) (in Russian)
- [ZD03] Zinder, Y., Do, V.H.: Scheduling with two criteria. In: Kendall, G., Burke, E., Petrovic, S. (ed) *Proceedings of The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, The University of Nottingham, 468–472 (2003)
- [ZR98] Zinder, Y., Roper, D.: An iterative algorithm for scheduling unit-time operations with precedence constraints to minimise the maximum lateness. *Annals of Operations Research*, **81**, 321–340 (1998)

---

# The Worst-case Analysis of the Garey-Johnson Algorithm for preemptive tasks on $m$ processors

Claire Hanen<sup>1</sup> and Yakov Zinder<sup>2</sup>

<sup>1</sup> LIP6, Université Pierre et Marie Curie, Paris, France, [claire.hanen@lip6.fr](mailto:claire.hanen@lip6.fr)

<sup>2</sup> Department of Mathematics, Faculty of Science, University of Technology,  
Sydney, Australia, [yakov.zinder@uts.edu.au](mailto:yakov.zinder@uts.edu.au)

**Summary.** The classical Garey-Johnson algorithm was originally developed for the maximum lateness problem with two parallel identical machines and partially ordered unit execution time tasks, but the main idea of the Garey-Johnson algorithm is applicable far beyond the confines of this particular problem. This paper analyzes the worst-case performance of a generalization of the Garey-Johnson algorithm for the problem with arbitrary number of machines and partially ordered tasks allowing preemptions. The presented performance guarantee is the best among all currently known for this problem.

**Key words:** machine scheduling, parallel machines, precedence constraints, worst-case analysis, maximum lateness

## 1 The problem

The maximum lateness scheduling problem, considered in this paper, can be stated as follows. A set  $N = \{1, 2, \dots, n\}$  of  $n$  tasks (jobs, operations) is to be processed on  $m > 1$  identical parallel machines (processors) subject to precedence constraints in the form of an anti-reflexive, anti-symmetric and transitive relation on  $N$ . If task  $i$  precedes task  $j$ , denoted  $i \rightarrow j$ , then the processing of  $i$  must be completed before the processing of  $j$  begins. All machines are available from time  $t = 0$ . Each machine can process at most one task at a time, and each task can be processed by any machine. The processing time of task  $j$  is denoted by  $p_j$  and can be any positive number. The processing of any task can be interrupted at any time and resumed later on the same or another machine, i.e. the preemptions are allowed.

A schedule  $\sigma$  is a vector function  $\sigma(t) = (\sigma_1(t), \dots, \sigma_n(t))$ , where each  $\sigma_j(t)$  is a continuous from the left piece-wise constant function, which has a finite number of points of discontinuity, and is equal to 1 if task  $j$  is processed at time  $t$ , and is equal to 0 otherwise. The completion time of task  $j$  in schedule  $\sigma$  is denoted by  $C_j(\sigma)$ . It is necessary to find a schedule, which minimizes the criterion of maximum lateness

$$L_{max}(\sigma) = \max_{j \in N} [C_j(\sigma) - d_j],$$

where  $d_j$  is a given due date associated with task  $j$ .

This problem is usually denoted by  $P|prec, pmtn|L_{max}$  (see for example [Br01]), where  $P$  specifies that the tasks are to be processed on several parallel identical machines,  $prec$  indicates the presence of precedence constraints, and the term  $pmtn$  indicates that the preemptions are allowed.

The non-preemptive counterpart of the  $P|prec, pmtn|L_{max}$  problem is the problem  $P|prec, p_j = 1|L_{max}$ , which differs from  $P|prec, pmtn|L_{max}$  only by two assumptions: all tasks have the same processing time of one time unit (indicated by the term  $p_j = 1$ ) and the preemptions are not allowed, i.e. if a machine begins executing a task, then it continues processing until its completion. If all due dates are equal to zero, then  $P|prec, pmtn|L_{max}$  and  $P|prec, p_j = 1|L_{max}$  convert to the makespan problems  $P|prec, pmtn|C_{max}$  and  $P|prec, p_j = 1|C_{max}$ , respectively, with the criterion

$$C_{max}(\sigma) = \max_{j \in N} C_j(\sigma).$$

## 2 The background and motivation

The  $P|prec, pmtn|C_{max}$  problem, and therefore the more general  $P|prec, pmtn|L_{max}$  problem, is NP-hard [Ull75]. The replacement of the makespan by the criterion of maximum lateness makes scheduling even more difficult. Thus, the replacement of  $C_{max}$  in the polynomially solvable  $P|out-tree, pmtn|C_{max}$  problem by  $L_{max}$  gives the NP-hard  $P|out-tree, pmtn|L_{max}$  (see the reference to [Len] in www). Here *out-tree* indicates that the precedence constraints are restricted to out-trees, i.e. partially ordered sets such that if  $i \rightarrow g$  and  $j \rightarrow g$ , then either  $i \rightarrow j$  or  $j \rightarrow i$ .

The complexity results inspired research, focused on various particular cases of the general problem, as well as the interest in approximation algorithms and their worst-case performance. One of the first results of this type was the Muntz-Coffman algorithm for the  $P2|prec, pmtn|C_{max}$  problem [MC70]. Another restricted version of the  $P|prec, pmtn|C_{max}$  problem was considered in [MQ05].

As far as the maximum lateness problem is concerned, the main source of ideas for models with preemptions has been various polynomial-time algorithms developed for  $P|prec, p_j = 1|L_{max}$  and its particular cases. One of such algorithms is the classical Garey-Johnson algorithm [GJ76], originally developed for  $P2|prec, p_j = 1|L_{max}$ , i.e. for the problem with two machines. Another well known algorithm is the algorithm presented in [BGJ77] for the  $P|in-tree, p_j = 1|L_{max}$  problem, where the term *in-tree* indicates that the precedence constraints are restricted to in-trees, i.e. the partially ordered sets such that if  $i \rightarrow j$  and  $i \rightarrow g$ , then either  $j \rightarrow g$  or  $g \rightarrow j$ . The preemptive counterparts of algorithms in [GJ76] and [BGJ77] have been introduced in [Law82] together with a modification of another Garey-Johnson algorithm [GJ77] originally developed for  $P2|prec, r_j, p_j = 1|L_{max}$ , i.e. for the problem where the processing of each task  $j$  can commence only after the given release time  $r_j$ . It was shown that the preemptive counterpart of the algorithm in [GJ76] solves  $P2|prec, pmtn|L_{max}$ , whereas a modification of the algorithm in [BGJ77] solves  $P|in-tree, pmtn|L_{max}$ . Another algorithm capable of solving both  $P2|prec, p_j = 1|L_{max}$  and  $P|in-tree, p_j = 1|L_{max}$  is presented in [ZR98]. As has been shown in [ZS], a generalization of the algorithm in [ZR98] solves both  $P2|prec, pmtn|L_{max}$  and  $P|in-tree, pmtn|L_{max}$ .

The worst-case performance of approximation algorithms for the problems with preemptions yet remains mostly unexplored area of machine scheduling. One of the first results in this area was a performance guarantee for the Muntz-Coffman algorithm [LS77]

$$\frac{C_{max}(\sigma^{MC})}{C_{max}(\sigma^*)} \leq 2 - \frac{2}{m},$$

where  $\sigma^{MC}$  is a schedule constructed by the Muntz-Coffman algorithm and  $\sigma^*$  is an optimal schedule for the  $P|prec, pmtn|C_{max}$  problem. Let the length of any path in the partially ordered set of tasks be the sum of completion times of tasks, constituting this path. As has been shown in [?],

$$L_{max}(\sigma^{BGJ}) - L_{max}(\sigma^*) \leq \frac{m-1}{m} \mathcal{L} - \min_{j \in N} p_j,$$

where  $\sigma^{BGJ}$  is a schedule constructed by the algorithm in [BGJ77],  $\sigma^*$  is an optimal schedule for the  $P|prec, pmtn|L_{max}$  problem, and  $\mathcal{L}$  is the length of the longest path in the partially ordered set of tasks.

Although the algorithm in [GJ76] was originally developed for the  $P2|prec, p_j = 1|L_{max}$  problem, it can be modified for the  $P|prec, pmtn|L_{max}$  problem.

Few years ago in [HZ03] we proved that:

$$L_{max}(\sigma^{GJ}) - L_{max}(\sigma^*) \leq \frac{m-2}{m} \mathcal{L},$$

In this paper, we derive a new bound on the performance ratio between  $L_{max}(\sigma^{GJ})$  and  $L_{max}(\sigma^*)$ , and show that this algorithm has the best known performance ratio for solving  $P|prec, pmtn|L_{max}$ . Moreover, this algorithm performs better than the Garey-Johnson algorithm for the non-preemptive counterpart  $P|prec, p_j = 1|L_{max}$  of the problem.

### 3 Consistent tails

In analysis of algorithms for the maximum lateness problem, it is often convenient to replace the original problem by an equivalent problem  $P|prec, pmtn|Q_{max}$ . In this problem, each task is associated with a value  $q_i$  called a tail, instead of a due date  $d_i$ . If  $\sigma$  is a schedule then we define  $Q_{max}(\sigma) = \max_{i \in T} C_i(\sigma) + q_i$ . Hence  $q_i$  represents the minimum amount of time that must be spent after the completion of task  $i$  till the end of the schedule  $Q_{max}(\sigma)$ .

It can be shown easily that if we consider an instance of  $P|prec, pmtn|L_{max}$  then by stating for each task  $i$ ,  $q_i = d_{max} - d_i$ , where  $d_{max} = \max_{i \in T} d_i$ ,  $Q_{max}$  minimization is equivalent to  $L_{max}$  minimization in the original instance.

Moreover, it appears that if we prove an approximation ratio for  $Q_{max}$ , say  $Q_{max}(\sigma) \leq \alpha Q_{max}(\sigma^*) - \beta$  then we can immediately state that  $L_{max}(\sigma) \leq \alpha L_{max}(\sigma^*) + (\alpha - 1)d_{max} - \beta$ .

Hence from now we shall consider problem  $P|prec, pmtn|Q_{max}$ .

For the sake of simplification, we shall also assume that to the problem is added a dummy task 0 with  $p_0 = 1$  and  $q_0 = 0$  that precedes every other task, and that will be processed in any schedule we consider at time  $-1$  (so that the other tasks can start at time 0).



The generalization of Garey-Johnson algorithm rely on several properties of the tails with respect to the  $Q_{max}$  criterium. Let us consider a task  $i$ , and a value  $q$ . We define  $S(i, q)$  to be the set of successors  $j$  of  $i$  such that  $p_j + q_j > q$ .

To each such set we associate a weight by measuring the minimum amount of the task that should be completed at time  $Q_{max}(\sigma) - q$  in any schedule  $\sigma$  as follows:

$$W(i, q) = \sum_{j \in S(i, q)} \min(p_j, p_j + q_j - q).$$

Then we prove the following lemma:

**Lemma 1** *In any schedule  $\sigma$ ,*

$$Q_{max}(\sigma) - C_i(\sigma) \geq q + \frac{W(i, q)}{m}$$

**Proof**

Let  $j$  be any task of  $S(i, q)$  and  $\sigma$  a schedule.  $C_j(\sigma) + q_j \leq Q_{max}(\sigma)$ . Moreover, we know that  $C_j(\sigma) \geq C_i(\sigma) + p_j$ . Assume that  $p_j \leq p_j + q_j - q$ , i.e  $q_j \leq q$ . then at time  $Q_{max} - q$  the task  $j$  should be completed. Otherwise  $C_j(\sigma) + q_j > Q_{max} - q + q_j \geq Q_{max}$ . If  $q_j > q$ , then at time  $Q_{max} - q$  at least  $p_j + q_j - q$  time units of task  $j$  have been processed. Otherwise at time  $C_j(\sigma) > Q_{max} - q + p_j - (p_j + q_j - q) = Q_{max} - q_j$ , the contradiction.

So, any task  $j \in S(i, q)$  must be processed during at least  $\min(p_j, p_j + q_j - q)$  time units during the interval  $(C_i(\sigma), Q_{max} - q]$ . On  $m$  machines, this last at least  $\frac{W(i, q)}{m}$  time units.  $\square$

This lemma states the idea of consistent tails. A set of tails is said to be consistent if for any task  $i$ , and for any rational value  $q$ ,

$$q_i \geq q + \frac{W(i, q)}{m}$$

The lemma 1 shows that we can, without loss of generality, move original tails of tasks to consistent ones. The basic algorithm to do that could be expressed as follows: while there is a task  $i$ , a value  $q$  such that  $q_i < q + \frac{W(i, q)}{m}$ , set  $q_i = q + \frac{W(i, q)}{m}$ .

As the values of  $q$  are not integers, this algorithm might not necessarily end. But, inspired by Lawler [Law82], by performing only useful iterations of the while loop, we provide a polynomial time algorithm to compute a set of consistent tails from original ones.

Consider at some step the values of  $p_j + q_j$  of successors  $j$  of a task  $i$  sorted by increasing order, let us denote them  $\gamma_1, \dots, \gamma_l$ . Then notice that if  $q, q' \in (\gamma_k, \gamma_{k+1}]$  then  $S(i, q) = S(i, q')$  although their weight might be different.

Let us consider a particular interval  $(\gamma_k, \gamma_{k+1}]$ . For  $q \in (\gamma_k, \gamma_{k+1}]$ , we define  $A(i, q) = \{j \in S(i, q), \quad q_j \leq q\}$ . Notice that :

$$\begin{aligned} W(i, q) &= \sum_{j \in A(i, q)} (p_j + q_j - q) + \sum_{j \in S(i, q) - A(i, q)} p_j \\ &= \sum_{j \in A(i, q)} (p_j + q_j) + \sum_{j \in S(i, q) - A(i, q)} p_j - q|A(i, q)| \end{aligned}$$

Let us notice  $\delta_1, \dots, \delta_i$  the values of  $q_j$  for  $j$  successor of  $i$  sorted by increasing order. Obviously, for  $q, q' \in (\delta_x, \delta_{x+1}] \cap (\gamma_k, \gamma_{k+1}]$ ,  $A(i, q) = A(i, q')$ . Let us now compare for such  $q \leq q'$  the two bounds  $q + \frac{W(i, q)}{m}$  and  $q' + \frac{W(i, q')}{m}$ .

$$q + \frac{W(i, q)}{m} - (q' + \frac{W(i, q')}{m}) = q - q' - (q - q') \frac{|A(i, q)|}{m}$$

From that we deduce that if  $|A(i, q)| > m$  then

$$q + \frac{W(i, q)}{m} \leq q' + \frac{W(i, q')}{m}$$

otherwise

$$q + \frac{W(i, q)}{m} \geq q' + \frac{W(i, q')}{m}$$

Hence again, the bound considered for  $q = \gamma_y$  or  $q = \delta_x$  dominates the other bounds.

From these properties we can derive a polynomial algorithm, that surely can be improved by the use of appropriate data structures:

Consider the tasks in reverse topological order. For each task  $i$ , sort the values  $p_j + q_j$  and  $q_j$  for  $j$  successor of  $i$  in increasing order.

Compute iteratively  $W(i, q)$  and  $q + \frac{W(i, q)}{m}$  for each such  $q$  value, and update  $q_i$ .

Without any particularly clever data structure, this can be done in  $O(n^2 \log n)$ .

In the rest of the paper, we shall denote by  $\mu_i$  the consistent tails of tasks computed from the original ones by this algorithm.

**Lemma 2** *In any schedule  $\sigma$ ,*

$$Q_{max}(\sigma) = \max_{i \in T} C_i(\sigma) + q_i = \max_{i \in T} C_i(\sigma) + \mu_i$$

$$Q_{max}^{opt} \geq \mu_0$$

**Proof**

This lemma is a simple outcome of lemma1.  $\square$

## 4 The GJ-algorithm with preemptions for an arbitrary number of processors

The algorithm considered in this paper is a generalization of that presented in [GJ77]. It also involves usual considerations about list scheduling with preemption, that have been presented in the litterature  $\square$

The basic idea is that the scheduling algorithm decomposes the schedule  $\sigma$  into so-called allocation intervals. Hence it will build from left to right an increasing sequence of dates, called allocation points,  $\lambda_{-1}, \lambda_0, \lambda_1, \dots, \lambda_z$ , where  $\lambda_{-1} = -1$  is the starting time of the dummy task 0,  $\lambda_0 = 0$  is its completion time.

The algorithm keeps track for each task  $i$  of its remaining processing time at time  $t$ , which is denoted by  $p_i(t, \sigma)$ . Initially, this value is equal to  $p_i$ .

At an allocation point  $t = \lambda_k$ , the algorithm considers the set  $F$  of tasks whose predecessors are completed before  $t$ , and whose remaining processing time is positive.

Let  $i$  be such a task. We call local priority of  $i$  at time  $t$  the quantity  $q_i(t, \sigma) = p_i(t, \sigma) + \mu_i$ .

If  $|F| \leq m$ , then each task of  $F$  is allocated to one machine and scheduled at time  $t$ , during  $\Delta = \min_{j \in F} (p_j(t, \sigma))$  time units. The next allocation point  $\lambda_{k+1} = t + \Delta$ , and the remaining processing times of tasks of  $F$  decrease by  $\Delta$ .

If  $|F| > m$ ,  $F$  is partitioned into several subsets  $F_1, \dots, F_x$  ( $x$  depends on  $t$ ), such that in each subset tasks have the same local priority. Moreover, if we take tasks  $j_l \in F_l$  then  $q_{j_1}(t, \sigma) > q_{j_2}(t, \sigma) > \dots > q_{j_x}(t, \sigma)$ .

Let  $y$  be the least index such that  $|F_1| + \dots + |F_y| \geq m$  (at least  $y \geq 1$ ). Each task of subsets  $F_1$  to  $F_y$  will perform partly in the next allocation interval. Notice that as the tasks of these subsets are independent to each other, hence if we know which portion of the tasks will be performed into the next interval, then Mac-Naughton algorithm can provide a schedule of these parts in an interval of given length  $\Delta$ , provided that the total workload is at most  $m\Delta$ .

The algorithm will determine as previously a length  $\Delta$  for the allocation interval, and for each task  $j \in F_1 \cup \dots \cup F_y$  a speed denoted by  $s_j(t) \leq 1$ , so that the part of task  $j$  performed in the interval will be of length  $s_j(t)\Delta$ . Moreover, we should have  $\sum_{j \in F_1 \cup \dots \cup F_y} s_j(t) = m$ , so that all portions of tasks can fit in an interval of length  $\Delta$  using the Mac-Naughton algorithm.

If  $j \in F_1 \cup \dots \cup F_{y-1}$  then  $s_j(t) = 1$  (the task is performed during the whole interval). Let  $M_y = m - (|F_1| + \dots + |F_{y-1}|)$ . If  $j \in F_y$ , we set  $s_j(t) = \frac{M_y}{|F_y|}$ .

Then the length of the interval  $\Delta$  is set to the maximum value so that for any task  $j \in F_1 \cup \dots \cup F_y$  the following conditions hold:

$$\begin{aligned} p_j(t + \Delta, \sigma) &= p_j(t, \sigma) - s_j(t)\Delta \geq 0 \quad (*) \\ \forall j' \in F \text{ such that } p_{j'}(t, \sigma) + \mu_{j'} &< p_j(t, \sigma) + \mu_j, \\ p_{j'}(t + \Delta, \sigma) + \mu_{j'} &\leq p_j(t + \Delta, \sigma) + \mu_j \quad (**) \end{aligned}$$

Then the next allocation point is set :  $\lambda_{k+1} = t + \Delta$ .

This way of constructing preemptive schedules using a priority based on remaining processing time plus a fixed latency has been mentioned first by Muntz and Coffman in [MC70]. It has been often used in the literature [LS95][Law82].

## 5 Properties of list schedules with tails

In, this section we prove some properties of the list-algorithm of the previous section. Notice that some of these properties do not depend on the initial properties of tails  $\mu_i$  (consistency), hence they can be used for the analysis of several kind of priorities.

For each allocation interval  $I_l = [\lambda_l, \lambda_{l+1}]$ , let us denote by

- $\hat{\mu}_l^L$  the maximum local priority of a feasible task at left point of the interval (i.e. at time  $\lambda_l$ ),
- $\rho_l$  the maximum speed of a task allocated to the interval.
- $\hat{\mu}_l^R$  the maximum local priority of a task performed in the allocation interval  $I_l$  at right point of the interval (i.e. at time  $\lambda_{l+1}$ ).
- $\hat{K}_l = |F_1(\lambda_l)|$  the number of tasks of maximum local priority performed in the interval.

- $\Delta_l = \lambda_{l+1} - \lambda_l$  the length of the interval.

Let us first remark that highest local priority at right point depend as follows on the highest priority at left point of an interval:

**Lemma 3** *If the priorities are consistent with precedence constraints, then for any integer  $-1 \leq l < z$ ,*

$$\hat{\mu}_l^L = \begin{cases} \hat{\mu}_l^R + \Delta_l & \text{If } \hat{K}_l \leq m \\ \hat{\mu}_l^R + \frac{\Delta_l * m}{\hat{K}_l} & \text{otherwise} \end{cases}$$

Moreover,  $\hat{\mu}_l^R \geq \hat{\mu}_{l+1}^L$ .

**Proof**

Notice that in both case, by property (\*\*) the tasks of the set  $F_1(\lambda_l)$  are still of highest priority at right point among tasks that were available at time  $\lambda_l$ . All tasks that become available at time  $\lambda_{l+1}$  are successors of tasks performed during the interval  $I_l$ . Hence the priority of any feasible task at time  $\lambda_{l+1}$  is less than or equal to  $\hat{\mu}_l^R$ .

Now, if  $\hat{K}_l \leq m$  tasks of  $F_1(\lambda_l)$  are performed at speed 1 so that a part of length  $\Delta_l$  of each of these tasks is performed, otherwise, the speed of these tasks is  $\frac{m}{\hat{K}_l}$ , and a part of length  $\frac{\Delta_l * m}{\hat{K}_l}$  of each is performed during the interval. Now let us consider a task  $j$  of maximum local priority in the next interval  $[\lambda_{l+1}, \lambda_{l+2})$ . If the local priority of  $j$   $q_j(\lambda_{l+1}, \sigma) > \hat{\mu}_l^R$  then  $j$  was not feasible at time  $\lambda_l$ , because the choice of  $\Delta_l$  in the algorithm implies that the priority order is not inverted at the end of the interval. Hence there should be a successor of a task  $i$  that ends during the interval  $[\lambda_l, \lambda_{l+1})$ . Now the consistency of tails implies that a successor  $j$  of a task  $i$  satisfies:  $p_j + \mu_j = q_j(\lambda_{l+1}, \sigma) \leq \mu_i = q_i(\lambda_{l+1}, \sigma)$ . The contradiction.  $\square$

Let us now consider the key property of list schedule in the preemptive case:

Let  $q$  be a positive number, and assume that in interval  $I_{k-1}$ , some task  $j$  with  $q_j(\lambda_k, \sigma) \geq q$  is allocated. Let us denote by  $o(k, q)$  the greatest integer less than  $k$  such that either a processor is idle in  $I_{o(k, q)}$  or there is at least one task  $i$  allocated in  $I_{o(k, q)}$  such that  $q_i(\lambda_{o(k, q)+1}, \sigma) < q$ . Notice that as dummy task 0 precedes every task, such an  $o(k, q)$  exists.

We also denote by  $M(k, q)$  the set of tasks partly performed during  $[\lambda_{o(k, q)+1}, \lambda_k)$ , to which we add the set of tasks partly performed during  $I_k$  the local priority of which is greater than or equal to  $q$  at time  $\lambda_{k+1}$ .

**Lemma 4** *Any task  $h \in M(k, q)$  either is partly performed or has a predecessor which is partly performed during the interval  $I_{o(k, q)}$ .*

**Proof**

Let  $h \in M(k, q)$ , and assume without loss of generality that  $h$  has no predecessor in  $M(k, q)$ . Assume that nor  $h$ , nor a predecessor of  $h$  is performed during the interval  $I_{o(k, q)}$ . Then during the iteration of the algorithm with current interval  $I_{o(k, q)}$ ,  $h$  belongs to the set of feasible tasks and has a lower priority at time  $\lambda_{o(k, q)}$  than

any other task performed during this interval (otherwise the algorithm would have scheduled a part of it), which should then perform at least  $m$  tasks. But at the end of this interval the order on local priorities is not inverted, so that  $q_h(\lambda_{o(k,q)+1}, \sigma) = q_h(\lambda_{o(k,q)}, \sigma) < q$ . But  $q_h(\lambda_{o(k,q)+1}, \sigma) \geq q_h(\lambda_k, \sigma) \geq q$ , the contradiction.  $\square$

Combining the two lemma leads to the following property:

**Lemma 5** *Let us consider a period  $\lambda_k$ , and set  $q = \hat{\mu}_k^L$ .*

$$\hat{\mu}_{o(k,q)}^R \geq \hat{\mu}_k^L + \frac{\lambda_k - \lambda_{o(k,q)+1}}{m-1}$$

**Proof**

Notice that in each interval  $I_x$ , with  $o(k, q) < x < k$ , all the performed tasks belong to  $M(k, q)$ , and that all processors perform such tasks during these periods. Let us denote, for a task  $h \in M(k, q)$ , the total amount of processing time spent in the interval  $[\lambda_{o(k,q)+1}, \lambda_k]$  by  $w_h = p_h(\lambda_{o(k,q)+1}, \sigma) - p_h(\lambda_k, \sigma)$ . Notice that if a part of  $h$  is performed during the interval  $I_{o(k,q)}$ , then  $q_h(\lambda_{o(k,q)+1}, \sigma) = q_h(\lambda_k, \sigma) + w_h > q$ .

Let us denote by  $X$  the set of tasks  $j$  performed in the interval  $I_{o(k,q)}$  with  $q_j(\lambda_{o(k,q)+1}, \sigma) \geq q$ . Let us denote by  $SM(j, q)$  the set of tasks in  $M(k, q)$  that are successors of  $j \in X$ . By lemma 4 any task of  $M(k, q)$  belongs to some set  $SM(j, q)$ , or is performed during the interval  $I_{o(k,q)}$ .

Notice that the set  $S(j, q) \subset SM(j, q)$ . Moreover, for each  $h \in SM(j, q)$ , obviously  $w_h \leq p_h$ . Now,  $p_h(\lambda_{o(k,q)+1}, \sigma) \leq p_h \leq p_h + p_h(\lambda_k, \sigma) + \mu_h - q$ . Thus  $w_h \leq p_h + \mu_h - q$ . Therefore  $w_h \leq \min(p_h, p_h + \mu_h - q)$ .

This implies that

$$\sum_{h \in SM(j, q)} w_h \leq W(j, q)$$

And we know that

$$\mu_j \geq q + \frac{W(j, q)}{m}$$

Consider now, for  $j \in X$ , the value  $w_j + \sum_{h \in SM(j, q)} w_h$ . We also know that

$$\sum_{j \in X} w_j + \sum_{h \in SM(j, q)} w_h = m * (\lambda_k - \lambda_{o(k,q)+1})$$

Notice that  $X$  contains at most  $m-1$  tasks, because if there is a task with lower local priority at right point of the interval, then according to the algorithm, all tasks with higher local priorities are performed at speed 1. Thus there is at least one term  $j$  of the previous sum for which

$$w_j + \sum_{h \in SM(j, q)} w_h \geq \frac{m * (\lambda_k - \lambda_{o(k,q)+1})}{m-1}$$

Now, let us consider  $q_j(\lambda_{o(k,q)+1}, \sigma) = w_j + p_j(\lambda_k, \sigma) + \mu_j$ .

$$\begin{aligned}
\hat{\mu}_{o(k,q)}^R &\geq q_j(\lambda_{o(k,q)+1}, \sigma) \geq w_j + p_j(\lambda_k, \sigma) + q + \frac{W(j, q)}{m} \\
&\geq \frac{w_j}{m} + q + \frac{\sum_{h \in SM(j, q)} w_h}{m} \geq q + \frac{\lambda_k - \lambda_{o(k,q)+1}}{m-1}
\end{aligned}$$

which achieves this proof.  $\square$

## 6 First decomposition of the schedule

In this section, we use the previous lemma to define a decomposition of the schedule that lead us in the next section to the worst case performance analysis of the Garey-Johnson schedule. Let  $\lambda_k$  be an allocation point, and let us define  $q = \hat{\mu}_{k-1}^R$  the number such that there is at least one task  $j$  that ends at time  $\lambda_k$  with remaining priority  $q_j(\lambda_k, \sigma) \geq q$ . Let us consider the following sequence of index  $Inc = \{k_0, \dots, k_z\}$ :  $k_0 = k$ ,  $k_x = o(k_{x-1}, q)$ , which is build until we reach the first index  $k_z$  such that in the interval  $I_{k_z}$  only one task has a local priority greater than  $q$  at time  $\lambda_{k_z+1}$ . Notice that such an interval will exist, since the dummy task is the only one performed in its interval and has maximum local priority, according to consistency of the tails. In the rest of the paper, the intervals  $I_{k_x}$  for  $0 < x < z$  will be referred to as *incomplete intervals*.

We denote by  $a(k)$  this index  $k_z$ , and by  $u(k)$  the task with local priority greater than  $q$  performed in interval  $I_{a(k)}$ .

Let us build the following set of tasks:

$$M = \cup_{x=0}^z M(k_x, q)$$

**Lemma 6**  $\forall j \in M, u(k)$  precedes  $j$ .

### Proof

$M$  equals the set of tasks performed in the interval  $[\lambda_{a(k)+1}, \lambda_k)$  which local priority at the right point of its allocation interval is at least  $q$ . Notice that according to lemma 4, for  $x < z$ , every task of  $M(k_x, q)$  has a predecessor or is performed in interval  $I_{k_{x+1}}$  and the local priority of this predecessor or task itself at time  $\lambda_{k_{x+1}}$  is therefore greater than  $q$ . Hence it will belong to the next set  $M(k_{x+1}, q)$  if  $x < z-1$ .

Thus any task in  $M$  has a predecessor or is performed in  $I_{a(k)}$ . And  $u(k)$  is the only task with local priority greater than  $q$  performed in this interval. Now, if  $a(k) + 1 \neq k$ , in interval  $I_{a(k)+1}$  at least two tasks are performed. One is a successor of  $u(k)$ . Hence  $u(k)$  must be completed at time  $\lambda_{a(k)+1}$ , and  $u(k)$  precedes every task of  $M$  if  $M$  is not empty.  $\square$

We denote, for each task  $h \in M$ , by  $w_h$  the amount of processing time of  $h$  performed in the interval  $[\lambda_{a(k)+1}, \lambda_k)$  while its local priority remains greater than  $q$ :  $w_h = p_h(\lambda_{a(k)+1}, \sigma) - p_h(\lambda_y, \sigma)$ , where  $y$  is the greatest integer such that  $a(k) < y \leq k$  with  $q_h(\lambda_y, \sigma) \geq q$ . We also denote by  $K_l$  the number of tasks of  $M$  performed in the interval  $I_l$ , with  $a(k) + 1 \leq l \leq k$ . Notice that for each incomplete interval  $I_{k_x}$  with  $0 < x < z$ ,  $K_{k_x} \geq 2$ . Notice also that in an interval  $I_l$  which is not incomplete, with  $k_z < l < k$ , all tasks  $j$  allocated to  $I_l$  satisfy  $q_j(\lambda_{l+1}, \sigma) \geq q$  and there are at least  $m$  such tasks ( $K_l \geq m$ ).

**Lemma 7**

$$\mu_{u(k)} \geq q + \sum_{x=1}^{z-1} \frac{2\Delta_{k_x}}{m} + \sum_{a(k) < l < k, l \notin Inc} \Delta_l$$

**Proof**

From lemma 6,  $u(k)$  precedes any task  $h$  in  $M$ , and  $h$  has a local priority at some right point greater than  $q$  hence  $p_h + \mu_h > q$ . This implies that  $h$  belongs to the set  $S(u(k), q)$ . Moreover, obviously  $w_h \leq p_h$ . Now,  $p_h(\lambda_{a(k)+1}, \sigma) \leq p_h \leq p_h + p_h(\lambda_y, \sigma) + \mu_h - q$  for  $y$  defined as the greatest integer such that  $p_h(\lambda_y, \sigma) + \mu_h \geq q$ . Thus  $p_h(\lambda_{a(k)+1}, \sigma) - p_h(\lambda_y, \sigma) = w_h$  and  $w_h \leq p_h + \mu_h - q$ . Therefore

$$w_h \leq \min(p_h, p_h + \mu_h - q)$$

This implies that  $W(u(k), q) \geq \sum_{h \in M} w_h$ .

On the other hand, let us denote by  $x_h(l)$  the boolean variable that is true if  $h$  is performed in interval  $I_l$  and if  $q_h(\lambda_{l+1}, \sigma) \geq q$ . Notice that  $w_h$  can be decomposed as follows:  $w_h = \sum_{k < l < a(k)} s_h(\lambda_l) \Delta_l x_h(l)$  because  $s_h(\lambda_l) \Delta_l$  equals the amount of time the schedules processes task  $h$  in interval  $I_l$  if it is processed. Hence:

$$\sum_{h \in M} w_h = \sum_{k < l < a(k)} \sum_{h \in M} s_h(\lambda_l) \Delta_l x_h(l)$$

Now, if  $I_l$  is an incomplete interval, then the speed of tasks of  $M$  is equal to one. As there are two such tasks at least, we can set that if  $l \in Inc$  then  $\sum_{h \in M} s_h(\lambda_l) \Delta_l x_h(l) \geq 2\Delta_l$ . If  $I_l$  is not an incomplete interval, only tasks of  $M$  processed in the interval, so that  $\sum_{h \in M} s_h(\lambda_l) x_h(l) = m$ . It follows that  $\sum_{h \in M} s_h(\lambda_l) \Delta_l x_h(l) \geq m\Delta_l$ . Therefore we get:

$$\sum_{h \in M} w_h \geq \sum_{l=1}^{z-1} 2\Delta_{k_x} + \sum_{a(k) < l < k, l \notin Inc} m\Delta_l$$

Now, from the definition of consistency, we know that

$$\mu_{u(k)} \geq q + \frac{W(u(k), q)}{m}. \text{ So, } \mu_{u(k)} \geq q + \frac{\sum_{h \in M} w_h}{m}. \quad \square$$

Let us now consider the  $K_{max}$  to be the maximum between  $m$  and the maximum number of tasks processed in a single interval of the schedule. From lemma 3 we deduce that

**Lemma 8**

$$\mu_{u(k)} \geq q + \sum_{l=1}^{z-1} \Delta_{k_x} + \sum_{a(k) < l < k, l \notin Inc} \frac{m\Delta_l}{K_{max}}$$

**Proof**

Indeed, according to lemma 3, if  $I_l$  is an incomplete interval, then  $\hat{\mu}_l^L = \hat{\mu}_l^R + \Delta_l$ , since then there will be less than  $m$  tasks in the set of tasks with maximum priority. Otherwise, if  $I_l$  is not an incomplete interval, then we can consider that  $\hat{\mu}_l^L \geq \hat{\mu}_l^R + \frac{m\Delta_l}{K_{max}}$ . Now, again according to lemma 3 for any time allocation point  $\lambda_l$ , we

have  $\hat{\mu}_{i-1}^R \leq \hat{\mu}_i^L$ . Now, starting from the inequality  $\hat{\mu}_{k-1}^R \geq q$ , and summing all these inequalities along the intervals, we get

$$\mu_{u(k)} = \hat{\mu}_a(k)^R \geq q + \sum_{l=1}^{z-1} \Delta_{k_x} + \sum_{a(k) < l < k, l \notin Inc} \frac{m \Delta_l}{K_{max}}$$

□

We shall derive another bound of the same kind based on lemma 5 which does not rely on this maximum number of tasks performed in parallel.

**Lemma 9**

$$\mu_{u(k)} \geq q + \sum_{l=1}^{z-1} \Delta_{k_x} + \sum_{a(k) < l < k, l \notin Inc} \frac{\Delta_l}{m-1}$$

**Proof**

Let us consider two incomplete intervals  $I_{k_x}$  and  $I_{k_{x+1}}$ . We will decompose again the schedule between these two intervals as follows: set  $q' = \hat{\mu}_{k_x-1}^R$ . Then consider  $s_1 = o(k_x, q')$ . Notice that as the incomplete interval  $I_{k_{x+1}}$  contains a task with priority at right point less than  $q < q'$ , so that necessarily  $s_1 \geq k_{x+1}$ . From lemma 5, we know that

$$\hat{\mu}_{s_1}^R \geq q' + \frac{\lambda_{k_x} - \lambda_{s_1+1}}{m-1}$$

Moreover  $\hat{\mu}_{s_1}^L = \hat{\mu}_{s_1}^R + \Delta_{s_1}$ . Hence if  $s_1 > k_{x+1}$ ,

$$\hat{\mu}_{s_1}^L \geq \hat{\mu}_{k_x-1}^R + \frac{\lambda_{k_x} - \lambda_{s_1}}{m-1}$$

Otherwise, we get

$$\hat{\mu}_{k_{x+1}}^R \geq \hat{\mu}_{k_x-1}^R + \frac{\lambda_{k_x} - \lambda_{k_{x+1}+1}}{m-1}$$

which can be rewritten as

$$\hat{\mu}_{k_{x+1}}^R \geq \hat{\mu}_{k_x-1}^R + \sum_{l=k_{x+1}+1}^{k_x-1} \frac{\Delta_l}{m-1}$$

Now assume that  $s_1 > k_{x+1}$ , if we set  $q' = \hat{\mu}_{s_1-1}^R \geq \hat{\mu}_{s_1}^L$ , we can build  $s_2 = o(s_1, q')$  and if  $s_2 > k_{x+1}$  we derive from lemma 5 the same kind of inequality:

$$\hat{\mu}_{s_2}^L \geq \hat{\mu}_{s_1-1}^R + \frac{\lambda_{k_x} - \lambda_{s_1}}{m-1}$$

and can iterate until this decomposition reaches  $s_y = k_{x+1}$  for some  $y$ . Then displaying along the inequalities will lead to :

$$\hat{\mu}_{k_{x+1}}^R \geq \hat{\mu}_{k_x-1}^R + \sum_{l=k_{x+1}+1}^{k_x-1} \frac{\Delta_l}{m-1}$$



Now, according to lemma 3,  $\hat{\mu}_{k_x}^L = \Delta_{k_x} + \hat{\mu}_{k_x}^R$  for any incomplete interval  $k_x$ . Hence from these inequalities, we derive :

$$\mu_{u(k)} = \hat{\mu}_{a(k)} \geq \hat{\mu}_{k_0-1}^R + \sum_{l=1}^{z-1} \Delta_{k_x} + \sum_{a(k) < l < k, l \notin Inc} \frac{\Delta_l}{m-1}$$

As  $\hat{\mu}_{k_0-1}^R = q$ , the lemma is proved.  $\square$

Let us denote by  $y_l$  the boolean variable which is true if  $I_l$  is an incomplete interval, false otherwise. The inequalities of lemma 7, 8 and 9 can be rewritten as follows:

$$\begin{aligned} \hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R &\geq \sum_{a(k) < l < k} \frac{2\Delta_l}{m} y_l + \Delta_l (1 - y_l) \\ \hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R &\geq \sum_{a(k) < l < k} \Delta_l y_l + \frac{m\Delta_l}{K_{max}} (1 - y_l) \\ \hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R &\geq \sum_{a(k) < l < k} \Delta_l y_l + \frac{\Delta_l}{m-1} (1 - y_l) \end{aligned}$$

From lemma 7 and 8 we now deduce the following result:

**Lemma 10**

$$\lambda_k - \lambda_{a(k)+1} \leq (1 + (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2}) (\hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R)$$

**Proof**

Let us consider  $\alpha$  and  $\beta$  two numbers such that

$$\frac{2}{m} \alpha + \beta = 1 \quad (1)$$

$$\alpha + \frac{m}{K_{max}} \beta = 1 \quad (2)$$

If we add the inequalities of lemma 7 and 8 multiplying the first one by  $\alpha$  and the second one by  $\beta$ , we get:

$$(\alpha + \beta)(\hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R) \geq \sum_{a(k) < l < k} (\frac{2}{m} \alpha + \beta) \Delta_l y_l + (\alpha + \frac{1}{K_{max}} \beta) \Delta_l (1 - y_l)$$

Thus

$$(\alpha + \beta)(\hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R) \geq \sum_{a(k) < l < k} \Delta_l = \lambda_k - \lambda_{a(k)+1}$$

Now, it is easy to see that solving the system on  $\alpha$  and  $\beta$  will lead to  $\alpha = \frac{K_{max} - m}{K_{max} - 2}$ ,  $\beta = 1 - \frac{2}{m} \alpha$ , so that  $\alpha + \beta = 1 + (1 - \frac{2}{m}) \alpha$ .  $\square$

Notice that  $\frac{K_{max} - m}{K_{max} - 2}$  is an increasing function of  $K_{max} \leq n$ , and that if  $K_{max} = m$ , then

$$\lambda_k - \lambda_{a(k)} \leq \hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R$$

From lemma 7 and 9 we deduce a second bound:

**Lemma 11**

$$\lambda_k - \lambda_{a(k)+1} \leq (2 - \frac{3}{m+1})(\hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R)$$

**Proof**

We use the same technique as in the previous lemma, by defining  $\alpha$  and  $\beta$  such that:

$$\frac{2}{m}\alpha + \beta = 1 \quad (3)$$

$$\alpha + \frac{1}{m-1}\beta = 1 \quad (4)$$

Summing the two inequalities of lemma 7 and 9 will lead to

$$(\alpha + \beta)(\hat{\mu}_{a(k)}^R - \hat{\mu}_{k-1}^R) \geq \sum_{a(k) < l < k} \Delta_l = \lambda_k - \lambda_{a(k)+1}$$

We can then calculate  $\alpha = \frac{m(m-2)}{m(m-1)-2} = \frac{m}{m+1}$ , and thus

$$\alpha + \beta = 1 + (1 - \frac{2}{m})\alpha = 1 + \frac{m-2}{m+1} = 2 - \frac{3}{m+1}$$

□

## 7 Second decomposition of the schedule

In this section we determine two bounds on the worst case performance ration of the Garey-Johnson algorithm, based on lemma 10 and 11.

First let us consider a task  $j$  such that  $C_j(\sigma) + \mu_j = Q_{max}(\sigma)$ .

**Lemma 12** *There exists an allocation interval  $I_l$  such that  $\lambda_{l+1} + \hat{\mu}_l^R = Q_{max}(\sigma)$*

**Proof**

Let  $I_l$  be the interval in which  $j$  ends. As  $q_j(\lambda_{l+1}, \sigma) = \mu_j \leq \hat{\mu}_l^R$  and as  $C_j(\sigma) \leq \lambda_{l+1}$ , we get  $Q_{max}(\sigma) \leq \lambda_{l+1} + \hat{\mu}_l^R$ . Let us assume that this is a strict inequality. Let then  $i$  be a task such that  $q_i(\lambda_{l+1}, \sigma) = \hat{\mu}_l^R$ . As at end point  $\lambda_{l+1}$ , tasks  $i$  and  $j$  do not have the same local priority, and as the algorithm preserves order on priorities within an interval,  $i$  belong to the set of highest priority tasks, which contains less than  $m$  tasks, so that  $i$  is performed at speed 1, so that  $i$  cannot end before  $\lambda_{l+1}$ . Hence  $C_i(\sigma) + \mu_i \geq \lambda_{l+1} + q_i(\lambda_{l+1}, \sigma) > Q_{max}(\sigma)$ , the contradiction. □

We shall now consider the leftmost allocation interval  $I_{s_0}$  such that  $\lambda_{s_0+1} + \hat{\mu}_{s_0}^R = Q_{max}(\sigma)$ . We then build a sequence  $s_0, \dots, s_w$  according to the first decomposition as follows :  $s_k = a(s_{k-1})$  until we reach  $s_w = -1$  (the corresponding interval is the one in which dummy task is computed).

We then use this decomposition to prove the following theorems:

**Theorem 1.**

$$Q_{max}(\sigma) \leq (1 + (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2}) Q_{max}^{opt}$$

**Proof**

Applying lemma 10 to each term of the sequence, will give:

$$\lambda_{s_{k-1}} - \lambda_{s_k+1} \leq (1 + (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2}) (\hat{\mu}_{s_k}^R - \hat{\mu}_{s_{k-1}-1}^R)$$

for each  $0 < k \leq w$ . Then by lemma 3, we know that  $\mu_{s_{k-1}-1}^R \geq \mu \mu_{s_{k-1}}^L = \mu_{s_{k-1}}^R + \Delta_{s_{k-1}}$ . Therefore

$$\lambda_{s_{k-1}+1} - \lambda_{s_k-1} \leq \hat{\mu}_{s_{k-1}-1}^R - \hat{\mu}_{s_k-1}^R$$

Hence

$$\lambda_{s_{k-1}+1} - \lambda_{s_k+1} \leq (1 + (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2}) (\hat{\mu}_{s_k}^R - \hat{\mu}_{s_{k-1}}^R)$$

So, summing the inequalities leads to:

$$\lambda_{s_0+1} - \lambda_{s_w+1} \leq (1 + (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2}) (\hat{\mu}_{s_w}^R - \hat{\mu}_{s_0}^R)$$

So, as  $\lambda_{s_w+1} = 0$ ,  $\hat{\mu}_{s_w}^R = \mu_0 \leq Q_{max}^{opt}$  we get:

$$\lambda_{s_0+1} + \hat{\mu}_{s_0}^R \leq (1 + (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2}) Q_{max}^{opt} - (1 - \frac{2}{m}) \frac{K_{max} - m}{K_{max} - 2} \hat{\mu}_{s_0}^R$$

As  $\hat{\mu}_{s_0}^R \geq 0$ , we get the result.  $\square$

**Theorem 2.**

$$Q_{max}(\sigma) \leq (2 - \frac{3}{m+1}) Q_{max}^{opt}$$

**Proof**

The proof relies on the same kind of arguments as in theorem 1. Using the second decomposition, we claim that:

$$\lambda_{s_{k-1}+1} - \lambda_{s_k+1} \leq (2 - \frac{3}{m+1}) (\hat{\mu}_{s_k}^R - \hat{\mu}_{s_{k-1}}^R)$$

And then that:

$$\lambda_{s_0+1} + \hat{\mu}_{s_0}^R \leq (2 - \frac{3}{m+1}) Q_{max}^{opt} - (1 - \frac{3}{m+1}) \hat{\mu}_{s_0}^R$$

$\square$

Notice that if  $K_{max} = m$ , then the Garey-Johnson schedule is optimal according to theorem 1. We can observe that the bound of theorem 2 outperforms the bound of theorem 1 for  $K_{max} \geq m(m-1)$ .

Moreover the two bounds outperform the bound  $2 - \frac{2}{m}$  for even  $m$  (and  $2 - \frac{2}{m+1}$  for odd  $m$ ) proved in [HZ] of the Garey-Johnson algorithm for  $P|prec, p_i = 1|L_{max}$ , so that in the worst case performance the Garey-Johnson algorithm performs strictly better when preemption are allowed than for unit execution times.

## 8 Tightness

In this section we provide an example of a family of Out-forests,  $T_K$  on which we can prove that when  $K$  tends to infinity:  $Q_{max}(\sigma^{GJ}) \geq (2 - \frac{2}{\sqrt{m}+1})Q_{max}(\sigma^*)$ .

Although these examples does not achieve the bound of theorem 2, we can observe that 2 is asymptotically (when  $m$  tends to infinity) the worst case performance ratio of Garey-Johnson algorithm, which remained unknown for a long time.

Let us assume for the sake of simplicity that  $m$  is a square, i.e.  $\sqrt{m}$  is an integer. Otherwise, the previous bound can be approximated by the same technique using integral part of  $\sqrt{m}$ .

Let us consider a value  $l$ , which will be considered as a parameter, and let us define  $u = 1 - \frac{\sqrt{m}-1}{m}$ ,  $y = \frac{l}{\sqrt{m}-1}$ . Let  $K \geq 1$ . We also define the following sequences:

- $L_i = u^{i-1} * l$ ,  $i = 1, \dots, K$
- $X_1 = \sqrt{m}y$ , and  $X_i = \frac{m-\sqrt{m}}{m}L_{i-1}$  for  $i = 2, \dots, K+1$

Let us remark that the two sequences have the following properties, which will be used in the rest of the section:

$$\forall i \geq 2, L_i + \frac{X_i}{\sqrt{m}} = L_{i-1} \quad (5)$$

Let us define two other sequences:

- $\nu_1 = 0$ ,
- $\nu_i = \nu_{i-1} + L_{i-1} + \frac{1}{\sqrt{m}}X_{i-1} = \sum_{j=1}^{i-1} L_j + \frac{1}{\sqrt{m}} \sum_{j=1}^{i-1} X_j$ ,  $i = 1, \dots, K+1$
- $\mu_i = \nu_i + \frac{X_i}{\sqrt{m}}$ ,  $i = 1, \dots, K$

Notice that according to property 5,

$$\forall i \geq 2, \nu_i = L_1 + \frac{1}{\sqrt{m}}X_1 + \sum_{j=1}^{i-2} L_j = \frac{sqr{m}}{sqr{m}-1}l + \sum_{j=1}^{i-2} L_j \quad (6)$$

Let us define the out forest  $T_K$  as follows: Nodes of  $T_K$  are partitionned into two subsets called a-nodes and b-nodes. a-nodes of  $T_K$  are decomposed into  $K$  rows of  $\sqrt{m}$  nodes. Nodes of row  $i$  are denoted by  $a_1^i, \dots, a_{\sqrt{m}}^i$ . Their processing times equal  $L_i$  and each tail equals  $\mu_i$ . If  $i > 1$ , then  $a_j^i$  is a predecessor of  $a_j^{i-1}$ .

Each node  $a_j^i$  is the predecessor of  $m$  independent b-nodes, we shall denote by  $b_j^i(1), \dots, b_j^i(m)$ . These nodes have processing times all equal to  $\frac{X_i}{\sqrt{m}}$ . The tails of these b-nodes equals  $\nu_i$ .

To all these nodes are added  $m\sqrt{m}$  independent b-nodes, with processing time  $\frac{X_{K+1}}{\sqrt{m}}$  and tail  $\nu_{K+1}$ .

**Lemma 13** *The tails defined for  $T_K$  are consistent.*

**Proof**

As b-nodes are leafs of the tree, their tails are. Let us consider node  $a_1^i$ . The significant  $S(a_1^i, q)$  that should be considered are for  $q = \nu_k$  and for  $q = \mu_k$  for  $k < i$ . Let us observe that  $S(a_1^i, \nu_k)$  is composed of a-nodes  $a_1^k$  to  $a_1^{i-1}$  and b-nodes  $b_1^k(1), \dots, b_1^k(m)$  to  $b_1^i(1), \dots, b_1^i(m)$ . The weight of each task equals its processing time.

Hence  $W(a_1^i, \nu_k) = \sum_{j=k}^{i-1} L_j + m * \sum_{j=k}^i \frac{X_j}{\sqrt{m}}$ . Thus

$$\frac{W(a_1^i, \nu_k)}{m} \leq \mu_i - \nu_k$$

similarly,  $S(a_1^i, \mu_k)$  is composed of a-nodes  $a_1^k$  to  $a_1^{i-1}$  and b-nodes  $b_1^{k+1}(1), \dots, b_1^{k+1}(m)$  to  $b_1^i(1), \dots, b_1^i(m)$ . The weight of each task equals its processing time.

Hence  $W(a_1^i, \mu_k) = \sum_{j=k}^i -1L_j + m * \sum_{j=k+1}^i \frac{X_j}{\sqrt{m}}$ . Thus

$$\frac{W(a_1^i, \mu_k)}{m} \leq \mu_i - \mu_k$$

The same arguments can be used for other a-nodes.  $\square$

Let us now build the optimal schedule  $\sigma^*$  for this graph:

It is composed by  $K+1$  blocks  $D_{K+1}, \dots, D_1$  performed consecutively in this order. In block  $D_i$ ,  $i \geq 2$  are performed a-nodes  $a_1^{i-1}, \dots, a_1^1$  on the  $\sqrt{m}$  first processors, and b-nodes  $b_j^i(k)$  for  $j = 1, \dots, \sqrt{m}$  and  $k = 1, \dots, m$ . Such b-nodes have processing times  $\frac{X_i}{\sqrt{m}}$ , so that the workload for b-nodes equals  $mX_i$ . Notice that  $X_i = (\frac{m-\sqrt{m}}{m})L_{i-1}$  so that the workload of b-nodes in block  $D_i$  equals  $(m - \sqrt{m})L_{i-1}$  so that they can be scheduled in time length  $L_{i-1}$  on the  $m - \sqrt{m}$  remaining processors. Block  $D_1$  is composed with b-nodes  $b_j^1(k)$  for  $j = 1, \dots, \sqrt{m}$  and  $k = 1, \dots, m$ , and thus lasts  $X_1$ .

Hence the maximum tail in block  $D_i$  equals  $\nu_i$ , and the completion time of a b-node in block  $D_i$  is  $\sum_{j=i-1}^K L_j$ . Hence the objective for a node in  $D_i$  equals  $\nu_i + \sum_{j=i-1}^K L_j$ . By the definition of  $\nu_i$ , it is easy to see that this value is maximum for the first block  $D_{K+1}$ , so that  $Q_{max}(\sigma^*) = L_K + \nu_{K+1}$ . But by property 6,

$$Q_{max}(\sigma^*) = \sum_{j=1}^K L_j + \frac{sqr\sqrt{m}}{sqr\sqrt{m} - 1} l = \left( \frac{1 - u^K}{1 - u} + \frac{sqr\sqrt{m}}{sqr\sqrt{m} - 1} \right) l$$

Now let us consider the Garey-Johnson schedule. It is composed with A-blocks and B-blocks as follows: First, the tasks with maximum priority are performed, i.e. b-nodes  $b_j^{K+1}(k)$  for  $k = 1, \dots, m$  and  $j = 1, \dots, \sqrt{m}$ . This define the first B-block  $B_{K+1}$  which lasts  $X_{K+1}$ . Then the priority of a-nodes become competitive and a-nodes of row  $K$  are performed in the first A-block  $A_K$ , which lasts  $L_K$ . Then the b-nodes successors of tasks of block  $A_K$  become available, and there will be a new B-block  $B_K$  with duration  $X_K$ . a-nodes of row  $K-1$  become competitive with nodes of this block only after the end of the block. So that the Garey-Johnson schedules alternates B-blocks and A-blocks until the first B-block  $B_1$  composed with b-nodes  $b_j^1(k)$ . Completion time of a task of this last block  $B_1$  is equal to  $\sum_{i=1}^K L_i + \sum_{i=1}^{K+1} X_i$  and this is also the value  $Q_{max}(\sigma^{GJ})$ .

Hence by property 5,

$$Q_{max}(\sigma^{GJ}) = X_1 + \frac{2m - \sqrt{m}}{m} \sum_{j=1}^K L_j = \left( \frac{2m - \sqrt{m}}{m} \frac{1 - u^K}{1 - u} + \frac{sqr{tm}}{sqr{tm} - 1} \right) l$$

Let us now compute the ratio between the two schedules:

$$\frac{Q_{max}(\sigma^{GJ})}{Q_{max}(\sigma^*)} = \frac{\frac{2m - \sqrt{m}}{m}(1 - u^K) + \frac{sqr{tm}}{sqr{tm} - 1}(1 - u)}{(1 - u^K) + \frac{sqr{tm}}{sqr{tm} - 1}(1 - u)}$$

As  $\frac{sqr{tm}}{sqr{tm} - 1}(1 - u) = \frac{1}{\sqrt{m}}$ , and as  $u^K \rightarrow 0$  as  $K \rightarrow \infty$ , asymptotically this ratio equals:

$$\frac{\frac{2m - \sqrt{m}}{m} + \frac{1}{\sqrt{m}}}{1 + \frac{1}{\sqrt{m}}} = \frac{2\sqrt{m}}{\sqrt{m} + 1}$$

Hence

$$\frac{Q_{max}(\sigma^{GJ})}{Q_{max}(\sigma^*)} \rightarrow \left(2 - \frac{2}{\sqrt{m} + 1}\right)$$

when  $K \rightarrow \infty$ .

Now, notice that the ratio does not depend on parameter  $l$ , that should be chosen so that for fixed  $K$ , all durations and tails would be integral.

## 9 Conclusion

In this paper we generalized the Garey-Johnson algorithm to find solutions of the problem  $P|prec, pmtn|Q_{max}$ . We analysed its worst case behavior and provide two new bounds on the performance ratio. We provided an example that does not achieve tightly the second bound, but asymptotically has a ratio 2. We hope to prove that the second bound is tight or to improve the bounds in the very next future.

Now, it is well known that the problem with release times for two processors has been solved by Garey and Johnson in [GJ77]. The algorithm seems to be more efficient than the one developed without release times. But it seems that the computation of consistent tails, or modified due-dates, is much more complicated in the preemptive case.

In [LPP01], and [Z03] and [ZR98], several new algorithms have been defined for solving  $P|prec, p_i = 1|L_{max}$ , and to extend the processor context (using pipelined processors for example). It should be fruitful to see if these new ideas could be used in the context of preemptions, and to determine whether the performance ratio improves, as it does for Garey and Johnson algorithm.

## References

- [Br01] Brucker, P.: Scheduling algorithms, third edition. Springer, Berlin (2001)
- [BGJ77] Brucker, P., Garey, M.R., Johnson, D.S.: Scheduling equal-length tasks under tree-like precedence constraints to minimise maximum lateness. *Mathematics of Operations Research*. **2** 275–284 (1977)

- [GJ76] Garey, M. R., Johnson, D. S.: Scheduling tasks with nonuniform deadlines on two processors. *Journal of the Association for Computing Machinery*, **23**, 461–467 (1976)
- [GJ77] Garey, M. R., Johnson, D. S.: Two-processor scheduling with start-time and deadlines. *SIAM Journal on Computing*, **6**, 416–426 (1977)
- [HZ] Hanen, C., Zinder, Y.: The worst-case analysis of the Garey-Johnson algorithm. submitted for publication.
- [HZ03] Hanen, C., Zinder, Y.: Scheduling UET tasks and preemptive tasks on parallel processors under the restriction of precedence constraints. MAPSP, abstract (2003)
- [LS77] Lam, S., Sethi, R.: Worst case analysis of two scheduling algorithms. *SIAM J. Computing* **6**, 518–536 (1977)
- [Law82] Lawler, E.L.: Preemptive scheduling of precedence-constrained jobs on parallel machines. In: *Deterministic and stochastic scheduling*, Dempster M.A.H. et al. (ed), D. Reidel Publishing Company, 101–123 (1982)
- [Len] Lenstra, J.K.: unpublished manuscript.
- [LPP01] Leung, A., Palem, K.V., Pnueli, A.: Scheduling time-constrained Instructions on pipelined processors, *ACM transactions on programming languages and Systems*, **23**, 1, 73–103 (2001)
- [MQ05] Moukrim, A., Quilliot, A.: Optimal preemptive scheduling on a fixed number of identical parallel machine. *Operations Research Letters* **33**, 143–150 (2005)
- [MC70] Muntz, R.R., Coffman, E.G., Preemptive scheduling of real-time tasks on multiprocessor systems. *J. Assoc. Comput. Mach.*, **17** 324–338 (1970)
- [LS95] Z. Liu, E. Sanlaville: Preemptive Scheduling with Variable Profile, Precedence Constraints and Due Dates. *Discrete Applied Mathematics* **58** 253–280 (1995)
- [Ull75] Ullman, J.D.: NP-complete scheduling problems. *J. Comput. System Sci.* **10**, 384–393 (1975)
- [Ull76] J.D. Ullman: Complexity of sequencing problems. In J.L. Bruno, E.G. Coffman, Jr., R.L. Graham, W.H. Kohler, R. Sethi, K. Steiglitz, and J.D. Ullman, editors, *Computer and Job/Shop Scheduling Theory*. John Wiley and Sons Inc., New York, 1976.
- [ZS] Zinder, Y., Singh, G.: Preemptive scheduling on parallel processors with due dates. submitted for publication
- [ZR98] Zinder, Y., Roper, D.: An iterative algorithm for scheduling unit-time operations with precedence constraints to minimise the maximum lateness. *Annals of Operations Research*, **81**, 321–340 (1998)
- [Z03] Zinder, Y.: An Iterative Algorithm for Scheduling UET Tasks with Due Dates and Release Times. *European Journal of Operational Research* **149**, Issue 2, 1 404–416 (2003)

# SCHEDULING ORDERS IN DEDICATED AND FLEXIBLE MACHINE ENVIRONMENTS

Joseph Y-T. Leung<sup>1</sup>, Haibing Li<sup>1</sup>, and Michael Pinedo<sup>2</sup>

<sup>1</sup>*Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102*

<sup>2</sup>*Stern School of Business, New York University, New York, NY 10012*

**Abstract** We are interested in the problem of scheduling orders for different product types in a facility with a number of identical machines in parallel. Each order asks for certain amounts of various different product types which can be produced concurrently. Each product type can be produced on a subset of the machines. Two extreme cases of machine environment are of interest. In the first case, each product type can be produced by one and only one dedicated machine. In the second case, all machines are flexible so that each product type can be produced by any one of machines. In addition, in this case, when a machine switches over from one product type to another, no setup is required. Each order has a release date and a weight. Preemptions are not allowed. The objective is minimizing the total weighted completion time of the orders. Even when all orders are available at time 0, both extreme cases have been shown to be NP-hard for any fixed number ( $\geq 2$ ) of machines. This paper focuses on the design and analysis of approximation algorithms for these two machine environment cases. We also present empirical comparisons of the various algorithms. The conclusions from the empirical analyses provide insights into the trade-offs with regard to solution quality, speed, memory space, and implementation complexity.

## 1. Introduction

In **order scheduling models**, we consider a facility with  $m$  parallel machines that can produce  $k$  different product types. Assume there are  $n$  orders from  $n$  different clients. Each order  $j = 1, 2, \dots, n$  requests a quantity of product type  $l = 1, 2, \dots, k$  which requires  $p_{lj} \geq 0$  units of processing. If  $p_{lj} = 0$ , it implies that order  $j$  does not request product type  $l$ . Each order  $j$  may also have a *release date*  $r_j$  which denotes the time the order arrives in, a *due date*  $d_j$  which represents the committed shipping date, and a weight  $w_j$  which ranks the importance of order  $j$ . Note that the product types of order  $j$  are allowed to be completed after its due date, but then a penalty is incurred. When a due date *must* be met, it is referred to as a *deadline* and denoted by  $\bar{d}_j$ . In certain environments, there may exist *precedence* constraints that require one or more orders be completed before another order is allowed to start its processing.

For any order  $j$ , each product type  $l = 1, 2, \dots, k$  can be produced on a subset of the  $m$  machines, namely  $M_l \subseteq \{1, 2, \dots, m\}$ . The  $m$  machines can be *identical*, *uniform*, or *unrelated*. In an identical machine environment, a product type  $l$  of order  $j$  requires  $p_{lij} = p_{lj}$  units of processing time on any machine  $i \in M_l$  if each machine is assumed to have a speed of 1. In a uniform machine environment, each machine  $i$  has a speed  $v_i > 0$ , and thus a product type  $l$  of order  $j$ , if entirely processed on a machine  $i \in M_l$ , would take  $p_{lij} = p_{lj}/v_i$  units of time to process. Finally, in an unrelated machine environment, the machines have different capabilities and performance on processing a product type  $l$ . In other words, the speed of a machine  $i \in M_l$  on product type  $l$ ,  $v_{li}$ , is dependent on both the machine and product type; product type  $l$  of order  $j$  requires  $p_{lij} = p_{lj}/v_{li}$  units of processing time on machine  $i$ .



The parallel machine environment can be either *nonpreemptive* or *preemptive*. In a nonpreemptive environment, each product type of an order must be processed in an uninterrupted fashion on a machine. In contrast, in some machine environments, preemptions are allowed, that is, a product type of an order can be processed for a period of time, interrupted and continued at a later point in time either on the same machine or on another machine. In the order scheduling models, it is assumed that preemptions disallow multiple machines to produce the same product for one order at the same time.

Product of the same type  $l$  but of different orders can be processed on a machine  $i \in M_l$  in a *batch*, without setup time being incurred between orders. However, prior to processing the whole batch of type  $l$ , a setup time may incur. When a batch of type  $l$  is immediately preceded by a different batch of type  $l'$ , the setup time on machine  $i$  is denoted by  $s_{il'l}$ . If  $l$  is the first product type on machine  $i$ , then the setup time is denoted by  $s_{i0l}$ . If for each  $l$ ,  $s_{il'l} = s_{i0l} = s_{il}$ , then the setup times on machine  $i$  are *sequence independent*; otherwise, they are *sequence dependent*. If for each machine  $i$ ,  $s_{il'l} = s_{l'l}$  for all  $l'$  and  $l$  including the case  $l' = 0$ , then the setup times are *machine independent*; otherwise, they are *machine dependent*. If the setup times are both sequence independent and machine independent, then  $s_{il'l} = s_{i0l} = s_{il} = s_l$  for each machine  $i$ . Furthermore, if the setup times are sequence independent, machine independent and *product type independent*, then  $s_{il'l} = s_{i0l} = s_{il} = s_l = s$  for each machine  $i$ . When  $s = 0$ , no setup time is considered.

The objectives to be optimized are always a function of the completion time of the orders, which are dependent on the schedule. The *completion time* of the order  $j$ , denoted by  $C_j$ , is the time when the latest product type of this order is finished on one of the machines. Let  $C_{lj}$  denote the individual completion time of product type  $l$  of order  $j$  on one machine, it is obvious that  $C_j = \max_l \{C_{lj}\}$ .

The objectives may also be functions of due dates. The *lateness* of order  $j$  is defined as  $L_j = C_j - d_j$ . Note that it is positive when order  $j$  is completed later than its due date and negative when it is completed early. The *tardiness* of order  $j$  is defined as  $T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}$ . Thus, tardiness is always non-negative. The *unit penalty* of order  $j$  is defined as

$$U_j = \begin{cases} 1, & \text{if } C_j > d_j; \\ 0, & \text{otherwise.} \end{cases}$$

Several objectives are of interest, namely, the *makespan*  $C_{\max} \equiv \max_{1 \leq j \leq n} \{C_j\}$ , the *maximum lateness*  $L_{\max} \equiv \max_{1 \leq j \leq n} \{L_j\}$ , the *total weighted completion time*  $\sum w_j C_j$ , the *total weighted tardiness*  $\sum w_j T_j$ , and the *total weighted number of late orders*  $\sum w_j U_j$ .

The class of models described above are very rich. The two extreme cases that are of interest are:

- (i) The *fully dedicated* case: There are  $m$  machines and  $m$  product types; each machine  $i = 1, 2, \dots, m$  can produce only one type of the  $m$  product types. In other words, there exists a one-to-one mapping between the machine set and the product set. This implies that  $k = m$ ; for each  $l = 1, 2, \dots, k$ ,  $|M_l| = 1$  and  $M_l \neq M_{l'}$  if  $l \neq l'$ .
- (ii) The *fully flexible* case: There are  $m$  machines and each machine is capable of producing all  $k$  products, that is, for each product type  $l = 1, 2, \dots, k$ ,  $M_l = \{1, 2, \dots, m\}$ .

Of course, there are many models in between these two extreme cases. It should be noted that, for the fully dedicated case, since each machine is dedicated to produce only one product type, the machine environment types (identical, uniform, and unrelated) do not make much sense. Therefore, for this case, machine environment types are not differentiated.

The order scheduling models described above have many applications in various fields. For details, the reader is referred to Leung, Li, and Pinedo (2005a).

In this paper, we consider the two extreme cases of order scheduling, namely, the fully dedicated case and the fully flexible case. In particular, we assume in both cases that all  $m$  machines are

identical; each order may have a release date and a weight; preemptions are not allowed. For the fully flexible case, we also assume that no setup is required when a machine switches over from one product type to another. We focus on the objective of minimizing the total weighted completion time of the orders, i.e.,  $\sum w_j C_j$ .

Even when all orders are available at time 0, both cases have been shown to be NP-hard for any fixed number ( $\geq 2$ ) of machines (Sung and Yoon, 1998; Blocher and Chhajed, 1996). We focus on the design and analysis of approximation algorithms for these two machine environment cases. Due to space constraints, we omit the proofs for our results. Interested readers can contact with us for the proofs.

The paper is organized as follows. In the next section, we consider the dedicated case. The flexible case is considered in Section 3. We draw some conclusions in the last section.

## 2. The Dedicated Machine Environment

As mentioned before, in the dedicated machine environment, each machine can produce one and only one product type. Without loss of generality, we assume that machine  $i$  is the only machine that can produce type  $i$  and type  $i$  is the only type that can be produced on machine  $i$  (otherwise, we can always relabel the machines and product types to make it so), so that the subscript  $i$  refers to a machine as well as to a product type. We refer  $p_{ij} = p_{iij}$  as the time required to process product type  $i$  of order  $j$  on machine  $i$ . For convenience, we denote the problem of minimizing  $\sum w_j C_j$  as  $PD \mid r_j \mid \sum w_j C_j$ . An overview of the past work on some special cases of this problem can be found in Leung, Li and Pinedo (2005a).

The algorithms we considered are of two types: priority rules (either static or dynamic) and LP-based algorithms. The priority rules are only applicable to  $PD \parallel \sum w_j C_j$ , i.e., when  $r_j = 0$  for all  $j$ .

### 2.1 Priority Rules for $PD \parallel \sum w_j C_j$

Let  $\Omega$  denote the set of unscheduled orders. Assuming a partial schedule  $\pi$ , we consider five greedy ways of selecting the next order  $j^* \in \Omega$  to be added to the partial schedule.

The Weighted Shortest Total Processing Time first (*WSTP*) rule schedules the orders in increasing order of  $\sum_{i=1}^m p_{ij}/w_j$ .

The Weighted Shortest Maximum Processing Time first (*WSMP*) rule schedules the orders in increasing order of  $\max_i \{p_{ij}\}/w_j$ .

The Weighted Smallest Maximum Completion Time first (*WSMC*) rule first sequences the orders on each machine,  $i = 1, 2, \dots, m$ , in increasing order of  $p_{ij}/w_j$ . (Note that the order sequences on the various machines may turn out to be different.) The rule then computes the completion time for order  $j$  as  $C'_j = \max_{i=1}^m \{C_{ij}\}$ . In its second pass, the rule schedules the orders in increasing order of  $C'_j$ .

The Weighted Shortest Processing Time first rule applied to the machine with the largest load (*WSPL*) functions as a dynamic priority rule that generates a sequence of orders one at a time, each time selecting as the next order the order  $j^* \in \Omega$  such that  $j^* = \arg \min_{j \in \Omega} \{p_{i^*j}/w_j\}$  where  $i^*$  is the machine with the largest workload under  $\pi$ .

The Weighted Earliest Completion Time first (*WECT*) rule selects as the next order  $j^*$  which satisfies  $j^* = \arg \min_{j \in \Omega} \{(C_j - C_k)/w_j\}$ , where  $C_k$  is the finish time of the order that was scheduled immediately before order  $j^*$ . Ties may be broken arbitrarily. After an order  $j^*$  has been selected by *WECT* to be included in the partial schedule a postprocessing procedure may be invoked. This postprocessing procedure interchanges order  $j^*$  with order  $k$  in case  $C_{j^*} \leq C_k$  in order to obtain a better (or at least not a worse) solution. Note that the case  $C_{j^*} \leq C_k$  occurs only when  $p_{i^*j^*} = 0$ , where  $i^*$  is the machine on which order  $k$  has, over all machines, the largest finish time. Assume that after the swap the order immediately before order  $j^*$  is order  $l$ . If  $C_{j^*} \leq C_l$ , we proceed with

an interchange of order  $j^*$  with order  $l$ . We continue with this postprocessing until  $C_{j^*}$  is larger than the completion time of the order that immediately precedes it. Note that after each swap, the finish time of  $j^*$  either decreases or remains unchanged, while the finish time of each order that is swapped with  $j^*$  remains unchanged. This is due to the fact that order  $j^*$  has zero processing time on the machine on which the swapped order has its largest finish time. Thus, the postprocessing, if any, produces a solution that is no worse than the one without postprocessing. Note that when applying the *WECT* rule, there may at times be ties. Since ties may be broken arbitrarily, the *WECT* rule could lead to various different schedules with different values of objective functions.

Through a sorting algorithm, both *WSTP* and *WSMP* can be implemented to run in  $O(mn + n \lg n)$  time, and *WSMC* can be implemented to run in  $O(mn \lg n)$  time. Both *WSPT* and *WECT* can be implemented in a rather straightforward manner to run in  $O(mn^2)$  time.

When  $m = 2$ , Sung and Yoon (1998) showed that both *WSTP* and *WSMC* have an approximation ratio of 2. Actually, Wang and Cheng (2003) showed that *WSTP*, *WSMP* and *WSMC* are all  $m$ -approximation algorithms for  $PDm \parallel \sum w_j C_j$ . As for *WSPL*, Leung, Li and Pinedo (2005b) showed that the algorithm is unbounded even when all  $w_j = 1$ . However, an empirical analysis showed that it performs well in practice for  $w_j = 1$ . The *WECT* algorithm is a generalization of the  $m$ -approximation *ECT* algorithm introduced in Leung, Li and Pinedo (2005b). We can obtain the following result:

**THEOREM 1** *For  $PD \parallel \sum w_j C_j$ , the worst-case bound of *WECT* is  $m$ .*

It would not be surprising that the priority rules may perform better when the processing times of each order are subject to constraints that ensure some form of regularity in the processing times. Sung and Yoon (1998) showed that for  $m = 2$  the performance ratio of *WSTP* can be reduced to  $3/2$  when the processing times satisfy  $(p_{1j} + p_{2j})/2 \geq |p_{1j} - p_{2j}|$  for each  $j = 1, 2, \dots, n$ . We can show tighter bounds for the rules when the processing times are subject to additional constraints.

**THEOREM 2** *For  $PD \parallel \sum w_j C_j$ , if*

$$\sum_{i=1}^m p_{ij}/m \geq \max_{1 \leq i \leq m} \{p_{ij}\} - \min_{1 \leq i \leq m} \{p_{ij}\}$$

*for each order  $j = 1, 2, \dots, n$ , then the worst-case bound of *WSTP* is  $2 - \frac{1}{m}$ , while both *WSMP* and *WECT* have a worst-case bound of*

$$\frac{1}{1 + \mathcal{H}(m) - \mathcal{H}(2m - 1)},$$

*where  $\mathcal{H}(k) \equiv 1 + \frac{1}{2} + \dots + \frac{1}{k}$  is the harmonic series.*

Note that

$$\lim_{m \rightarrow \infty} \frac{1}{1 + \mathcal{H}(m) - \mathcal{H}(2m - 1)} = \frac{1}{1 - \ln 2} \approx 3.259.$$

**THEOREM 3** *For  $PD \parallel \sum w_j C_j$ , if  $\max_{1 \leq i \leq m} \{p_{ij}\} \leq 3 \min_{1 \leq i \leq m} \{p_{ij}\}$  for each order  $j = 1, 2, \dots, n$ , then each of *WSTP*, *WSMP* and *WECT* has a worst-case bound of  $3 - \frac{6}{m+2}$ .*

## 2.2 LP-Based Algorithms for $PD \mid r_j \mid \sum w_j C_j$

In this subsection we allow orders to have different release dates. Preemptions are not allowed. However, unforced idleness of the machines is allowed. We present two approximation algorithms based on different LP relaxations.

**Algorithm Based on a Completion Time Formulation.** Hall et al. (1997) presented a 3-approximation LP-based algorithm for  $1 \mid r_j \mid \sum w_j C_j$ . We shall extend this algorithm to solve  $PD \mid r_j \mid \sum w_j C_j$ .

Let  $\mathcal{O} = \{1, 2, \dots, n\}$  denote the set of all orders. For any subset  $\mathcal{S} \subseteq \mathcal{O}$ , let  $p_i(\mathcal{S}) = \sum_{j \in \mathcal{S}} p_{ij}$ ,  $p_i^2(\mathcal{S}) = \sum_{j \in \mathcal{S}} p_{ij}^2$ ,  $i = 1, 2, \dots, m$ . The  $PD \mid r_j \mid \sum w_j C_j$  problem can be relaxed by the following linear program:

$$LP_1 = \text{minimize } \sum_{j=1}^n w_j C_j$$

subject to

$$C_j \geq r_j + p_{ij}, \quad i = 1, \dots, m, j = 1, \dots, n; \quad (1)$$

$$\sum_{j \in \mathcal{S}} p_{ij} C_j \geq \frac{p_i^2(\mathcal{S}) + (p_i(\mathcal{S}))^2}{2}, \quad i = 1, \dots, m, \text{ for each } \mathcal{S} \subseteq \mathcal{O}. \quad (2)$$

Constraints (1) are trivial. However, constraints (2) need some justification. Assume that  $\mathcal{S} = \{1, 2, \dots, |\mathcal{S}|\}$ . It follows that for  $j \in \mathcal{S}$ ,

$$C_j \geq C_{ij} \geq \sum_{k \leq j} p_{ik}, \quad i = 1, \dots, m.$$

The inequality is due to the fact that there may be some idle time in the schedule because of the release dates. Thus,

$$p_{ij} C_j \geq p_{ij} C_{ij} \geq p_{ij} \sum_{k \leq j} p_{ik}.$$

Summing  $p_{ij} C_j$  over all  $j \in \mathcal{S}$  and simple algebra results in (2).

It is clear that (2) generates an exponential number of constraints. For the one-machine case, Queyranne (1993) has shown that such constraints can be separated polynomially so that the above linear program can be solved in polynomial time by the ellipsoid method. This is the key observation that the above linear program can be used as a relaxation for approximation algorithms. Now consider the following algorithm:

#### An LP-based Algorithm Using Completion Times ( $H_{LP1}$ )

---

**Step 1:** Solve  $LP_1$  and let the optimal solution be  $\overline{C}_1, \overline{C}_2, \dots, \overline{C}_n$ .

**Step 2:** Schedule the orders in nondecreasing order of  $\overline{C}_j$ . Ties are broken arbitrarily. Insert idle time when  $r_j$  is greater than the completion time of the  $(j-1)^{th}$  order.

---

We can show the following results for the above LP-based algorithm:

**THEOREM 4** *The worst-case bound of  $H_{LP1}$  for solving  $PD \parallel \sum w_j C_j$  and  $PD \mid r_j \mid \sum w_j C_j$  is 2 and 3, respectively.*

**Algorithm Based on a Time Interval Formulation.** Inspired by the time interval indexed linear programming formulation for  $P \mid r_j \mid \sum w_j C_j$  due to Hall et al. (1997), Wang and Cheng (2003) presented a 16/3-approximation algorithm for  $PD \parallel \sum w_j C_j$ . In what follows, we present an extension of Wang and Cheng's algorithm for  $PD \mid r_j \mid \sum w_j C_j$ .

Given  $\lambda > 1$ , we divide the time horizon of potential completion times into the intervals:  $[1, 1], (1, \lambda], (\lambda, \lambda^2], \dots, (\lambda^{L-1}, \lambda^L]$ , where  $L$  is the smallest integer so that  $\lambda^L \geq \max_{1 \leq j \leq n} \{r_j\} +$

$\max_{1 \leq i \leq m} \{\sum_{j=1}^n p_{ij}\}$ . For convenience, let  $t_0 = 1$ , and  $t_l = \lambda^{l-1}, l = 1, \dots, L$ . Thus, the  $l^{th}$  interval runs from  $t_{l-1}$  to  $t_l, l = 1, 2, \dots, L$ . Let decision variable  $x_{jl}$  be:

$$x_{jl} = \begin{cases} 1, & \text{if order } j \text{ is scheduled to finish within the interval } (t_{l-1}, t_l]; \\ 0, & \text{otherwise.} \end{cases}$$

Consider the following linear programming relaxation:

$$LP_2 = \text{minimize } \sum_{j=1}^n w_j \sum_{l=1}^L t_{l-1} x_{jl}$$

subject to

$$\sum_{l=1}^L x_{jl} = 1, j = 1, \dots, n; \quad (3)$$

$$\sum_{k=1}^l \sum_{j=1}^n p_{ij} x_{jk} = \sum_{j=1}^n p_{ij} \sum_{k=1}^l x_{jk} \leq t_l, \quad i = 1, \dots, m, l = 1, \dots, L; \quad (4)$$

$$x_{jl} = 0, \text{ if } t_l < r_j + p_{ij}, \quad j = 1, \dots, n, l = 1, \dots, L; \quad (5)$$

$$x_{jl} \geq 0, \quad j = 1, \dots, n, l = 1, \dots, L. \quad (6)$$

The following algorithm is based on the above LP relaxation:

#### An LP-based Algorithm Using Time Intervals ( $H_{LP2}$ )

**Step 1:** Given  $\lambda$ , solve  $LP_2$  and let the optimal solution be  $\bar{x}_{jl}, j = 1, \dots, n, l = 1, \dots, L$ .

**Step 2:** Let  $\bar{C}_j = \sum_{l=1}^L t_{l-1} \bar{x}_{jl}, j = 1, \dots, n$ .

**Step 3:** Schedule the jobs in nondecreasing order of  $\bar{C}_j$ . Ties are broken arbitrarily. Insert idle time when  $r_j$  is greater than the completion time of the  $(j-1)^{th}$  order.

---

Wang and Cheng (2003) showed that, given  $\lambda = 2$ ,  $H_{LP2}$  is a  $16/3$ -approximation algorithm for  $PD \parallel \sum w_j C_j$ . When the orders have different release dates, we can show the following result:

**THEOREM 5** *Given  $\lambda = 2$ ,  $H_{LP2}$  is a  $\frac{19}{3}$ -approximation algorithm for the problem  $PD \mid r_j \mid \sum w_j C_j$ .*

For  $LP_2$ , it would be of interest to investigate if a smaller  $\lambda$  leads to a better performance. In what follows, we present an empirical analysis.

### 2.3 Empirical Analysis of the Algorithms

Since the priority rules apply only to  $PD \parallel \sum w_j C_j$ , we focus our experiments on the problem with all release dates equal to zero.

For each problem size with  $n = 20, 50, 100, 200$  orders and  $m = 2, 5, 10, 20$  machines, 30 instances are randomly generated using a factor called *order diversity*. The order diversity  $k$  is used to characterize the number of product types each order requires. The following three cases of order diversity are considered:

$k = 2$ : In problem instances 1 to 6 each order requests 2 different product types. We denote this group as  $G_1$ .

$k = m$ : In problem instances 7 to 24 each order requests the maximum number of different product types, namely  $m$ , i.e, the number of machines. However, these 18 instances are grouped into the following 3 subgroups:

For instances 7 to 12 in group  $G_2$ , the processing times of each order have no additional constraints.

For instances 13 to 18 in group  $G_3$ , the processing times of each order  $j$  are subject to  $\max_{1 \leq i \leq m} \{p_{ij}\} \leq 3 \min_{1 \leq i \leq m} \{p_{ij}\}$ .

For instances 19 to 24 in group  $G_4$ , the processing times of each order  $j$  are subject to  $\sum_{i=1}^m p_{ij}/m \geq \max_i \{p_{ij}\} - \min_i \{p_{ij}\}$ .

$k = r$ : In problem instances 25 to 30 in group  $G_5$  each order requests a random number ( $r$ ) of different product types;  $r$  is randomly generated from the uniform distribution  $[1, m]$ .

When the number of product types,  $l$ , for each order  $j$  is determined,  $l$  machines are chosen randomly. For each machine  $i$  that is selected, an integer processing time  $p_{ij}$  is generated from the uniform distribution  $[1, 100]$ . Note that for instances 13 to 24, the processing times of each order are generated in such a way that they satisfy the additional requirements. In addition to the generation of processing times, for each order  $j$ , a weight is randomly generated from the uniform distribution  $[1, 10]$ . In total,  $4 \times 4 \times 30 = 480$  instances are generated.

The algorithms are implemented in C++. We applied the GLPK 4.4 (Makhorin, 2004) callable library to solve the linear programs in the  $H_{LP2}$  algorithm. For the  $H_{LP1}$  algorithm, since we could not find a callable library for the ellipsoid method, we did not implement it. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer (Pentium III 900Mhz plus 384MB RAM). It should be noted that the time-interval LP-based algorithm needs a significant amount of virtual memory. For example, for a problem instance with  $n = 200$  and  $m = 20$ , when we let  $\lambda = 2^{1/4}$ , the total memory usage for the time-interval LP-based algorithm could reach 1GB.

In what follows, we study the performance of the algorithms in terms of two aspects: the frequencies at which they are the best, and comparison of their average costs and average running times. For convenience, we define the following notation:  $LP_2^1$  – the  $H_{LP2}$  algorithm with  $\lambda = 2^{1/4}$ ;  $LP_2^2$  – the  $H_{LP2}$  algorithm with  $\lambda = \sqrt{2}$ ;  $LP_2^3$  – the  $H_{LP2}$  algorithm with  $\lambda = 2$ .

Due to space constraints, we omit the tables for experimental results, which are available at <http://web.njit.edu/~leung/pd-sumwjcj/tables.pdf>.

We first compare the performance of the algorithms in terms of the frequencies at which they are the best. From the results, we have the following findings:

- For group  $G_1$  with  $k = 2$ ,  $LP_2^1$  and  $WECT$  have the best performance. A close study of the results reveals that for instances with small  $n$  and large  $m$ ,  $WECT$  outperforms  $LP_2^1$ . However, when  $n$  becomes larger,  $LP_2^1$  becomes significantly better than all other algorithms.
- For group  $G_2$ , for which  $k = m$  but there are no additional constraints on the properties of processing times,  $LP_2^1$  performs the best, and  $WECT$  the second best. However,  $LP_2^1$  becomes significantly better for instances with large  $n$ .
- $WECT$  and  $LP_2^1$  are the two best algorithms for group  $G_3$ . However, when  $n$  is small,  $WSTP$ ,  $WSMP$ ,  $WSMC$ , and  $WSPL$  may occasionally beat the other algorithms. For large  $n$ , the tendency is that  $WECT$  beats  $LP_2^1$ . The results show similar findings for  $G_4$ . These results are consistent with our theoretical analysis that the priority rules exhibit a better performance with additional constraints on the processing times.
- For group  $G_5$ , for which  $k = r$ ,  $LP_2^1$  performs the best, and  $WECT$  the second best. The tendency is that  $LP_2^1$  becomes significantly better with large  $n$ .

- The comparison among  $LP_2^1$ ,  $LP_2^2$ , and  $LP_2^3$  shows that smaller  $\lambda$  leads to better performance of the  $H_{LP2}$  algorithm, even though there are some exceptions.

Now we compare the performance of the algorithms in terms of comparisons of average costs and average running times. For convenience, we use the notation  $\prec$  to indicate that algorithm  $A$  is better than algorithm  $B$  if  $A \prec B$ . Furthermore, if algorithm  $A$  almost ties with algorithm  $B$ , we denote it as  $A \sim B$ .

- For each  $n$ , the percentage of each priority rule (except for  $WECT$ ) tends to increase when  $m$  increases. This is consistent with our previous theoretical analysis that the performance of each priority rule becomes worse when  $m$  becomes larger. By contrast, the  $H_{LP2}$  algorithm does not have such a relationship between its performance with the value of  $m$ . Secondly, the results also show that a smaller  $\lambda$  leads to a better performance of  $H_{LP2}$ .
- For group  $G_1$ ,  $WECT$  can beat  $LP_2^1$  when  $n \leq 50$ . However, when  $n \geq 100$ ,  $LP_2^1$  is overwhelmingly better than other algorithms. Although it is hard to fix a ranking of performance when  $n$  is small, for large  $n$  we see that  $LP_2^1 \prec LP_2^2 \prec WECT \sim LP_2^3 \sim WSMC \prec WSTP \prec WSMP \prec WSPL$ .
- For group  $G_2$ ,  $WECT$  can beat  $LP_2^1$  when  $n \leq 20$ . However, when  $n \geq 50$ ,  $LP_2^1$  is much better than all other algorithms. For large  $n$  we see that  $LP_2^1 \prec WECT \prec LP_2^2 \prec LP_2^3 \prec WSTP \prec WSMC \prec WSMP \prec WSPL$ . Note that the difference between  $WECT$  and  $LP_2^1$  is almost less than 1%. Thus, the performance of  $WECT$  is actually quite close to that of  $LP_2^1$ . However, the results show that  $LP_2^1$  requires hours of running time for large instances, while  $WECT$  requires only milliseconds. As stated before, the  $H_{LP2}$  algorithm requires virtual memory up to 1GB. By contrast,  $WECT$  only requires several kilobytes of memory.
- For group  $G_3$ ,  $WECT$  can beat  $LP_2^1$  when  $n \leq 20$ . However, when  $n \geq 50$ ,  $LP_2^1$  is overwhelmingly better than other algorithms. For large  $n$  we can see:  $LP_2^1 \prec WECT \prec LP_2^2 \prec WSTP \prec WSMC \prec WSMP \prec LP_2^3 \sim WSPL$ . Note that the difference between  $WECT$  and  $LP_2^1$  is less than 0.5%. Thus, the performance of  $WECT$  is almost the same as that of  $LP_2^1$ . However, to achieve such performance,  $LP_2^1$  requires more computational resources than  $WECT$ . It is also interesting to see that,  $WSTP$ ,  $WSMC$ , and  $WSMP$  perform better than  $LP_2^3$ .
- For group  $G_4$ ,  $WECT$  is the best. The algorithms are ranked as follows:  $WECT \prec LP_2^1 \prec LP_2^2 \prec WSTP \prec WSMC \prec WSMP \prec LP_2^3 \sim WSPL$ . Again,  $WSTP$ ,  $WSMC$ , and  $WSMP$  perform better than  $LP_2^3$ .  $LP_2^1$  could not beat  $WECT$  even though it requires more computational resources.
- For group  $G_5$ ,  $LP_2^1$  is the best. The algorithms are ranked as:  $LP_2^1 \prec LP_2^2 \prec WECT \prec LP_2^3 \prec WSTP \prec WSMC \prec WSMP \prec WSPL$ . Again, the LP-based algorithm requires more computational resources than  $WECT$ .

To observe the cost-effective performance of the  $H_{LP2}$ , we compare for each group of instances the percentages that the average costs of  $LP_2^2$  and  $LP_2^3$  are larger than that of  $LP_2^1$ . From the results, we can see that the gap between the average cost of  $LP_2^2$  and that of  $LP_2^1$  is actually very small. For most cases, it is less than 1.0%; and the largest one is 1.3%. Therefore, the performance of  $LP_2^2$  is actually very close to that of  $LP_2^1$ . However, the average running time of  $LP_2^1$  is much more than that of  $LP_2^2$  (for some cases, it is more than 10 times). In addition, in the experiments, we noticed that the memory requirement of  $LP_2^1$  is twice that of  $LP_2^2$ . Thus, in practice, if the use

of  $H_{LP2}$  is considered, it is recommended to choose  $\lambda = \sqrt{2}$  in order to strike a balance between performance and the use of computational resources.

Summarizing the empirical analysis, we recommend the use of  $WECT$  and  $H_{LP2}$  with  $\lambda = \sqrt{2}$ . Especially, in an environment which requires a solution to be generated quickly with limited memory spaces,  $WECT$  is preferable. It is simple to implement, requires a small amount of memory, runs fast, and produces good results.

### 3. The Flexible Machine Environment

In the flexible case, each product type can be produced by any one of identical machines. We refer  $p_{lj} = p_{lij}$  as the time required to process product type  $l$  of order  $j$  on any machine  $i = 1, 2, \dots, m$ . When a machine switches over from one product type to another, we assume that no setup is required. For convenience, we denote the problem of minimizing  $\sum w_j C_j$  as  $PF \mid \Pi k \mid \sum w_j C_j$ , where  $\Pi k$  means that the facility can produce  $k$  different product types.

Since  $PF \mid \Pi k \mid \sum C_j$  is strongly NP-hard (Blocher and Chhajed, 1996), it follows that the more general version  $PF \mid \Pi k \mid \sum w_j C_j$  is NP-hard as well. We are interested in designing heuristics for  $PF \mid \Pi k \mid \sum w_j C_j$  that consist of two phases. The first phase determines the sequence of the orders, while the second phase assigns the individual jobs within each order to the specific machines. Based on this idea, we consider two classes of heuristics: *sequential two-phase heuristics* and *dynamic two-phase heuristics*.

#### 3.1 Sequential two-phase heuristics

The first phase of the *sequential two-phase heuristics* sequences the orders; the second phase assigns the individual jobs of each order to the specific machines. Rules for sequencing the orders include:

- The *Weighted Shortest Total Processing time first (WSTP)* rule which sequences the orders in increasing order of  $\sum_{l=1}^k p_{lj}/w_j$ .
- The *Weighted Shortest LPT Makespan first (WSLM)* rule which sequences the orders in increasing order of  $C_{LPT}^{(j)}/w_j, j = 1, 2, \dots, n$ , where  $C_{LPT}^{(j)}$  is the makespan of the schedule obtained by scheduling the jobs of order  $j$  on all  $m$  parallel machines according to the *longest processing time first (LPT)* rule, assuming each machine is available from time zero on.
- The *Weighted Shortest MULTIFIT Makespan first (WSMM)* rule which sequences the orders in increasing order of  $C_{MF}^{(j)}/w_j, j = 1, 2, \dots, n$ , where  $C_{MF}^{(j)}$  is the makespan of the schedule obtained by scheduling the jobs of order  $j$  on all  $m$  parallel machines according to the *MF* assignment rule which is described below, assuming each machine is available from time zero on.

After the sequence of the orders has been determined by one of the above rules, the individual jobs for each order are assigned to the specific machines according to one of the assignment rules listed below:

- The *List Scheduling* rule (*LS*) assigns in each iteration an unassigned (arbitrary) job to a machine with the smallest workload, until all jobs are assigned.
- The *Longest Processing Time first* rule (*LPT*) assigns in each iteration an unassigned job with the longest processing time to a machine with the smallest workload, until all jobs are assigned.
- The *Bin Packing* rule (*BIN*) first determines a target completion time for an order using the *LPT* assignment rule (just as a trial, not as a real assignment). This completion time



is used as a target completion time (bin size). At each iteration, the *BIN* rule assigns an unassigned job with the longest processing time to a machine with the largest workload. If the workload of the machine exceeds the target completion time after the assignment, then undo this assignment and try the assignment on the machine with the second largest workload. This try-and-check procedure is repeated until the job can be assigned to a machine without exceeding the target completion time. If assigning the job to the machine with the smallest workload still exceeds the target completion time, then assign it to this machine, and reset the target completion time as the completion time of the job on this machine. The whole procedure is repeated until all jobs are assigned to the machines.

- The *MULTIFIT* rule (*MF*) assigns the jobs of an order to the machines following an idea that is similar to (but not exactly the same as) the *MULTIFIT* algorithm for  $Pm \parallel C_{max}$  (see Coffman et al., 1978). The original *MULTIFIT* algorithm uses the *First Fit Decreasing* (*FFD*) rule for the bin packing problem. In contrast, we use the *Best Fit Decreasing* (*BFD*) rule. Let  $j$  be the order whose jobs are to be assigned. In the *BFD* procedure, we treat the machines as bins that are partially filled, and treat the jobs of order  $j$  as items whose sizes are exactly equal to their processing times. The jobs of order  $j$  are pre-sorted in nonincreasing order of their processing times. Given the partial schedule generated for the orders scheduled before order  $j$ , and given a target completion time  $t$  (bin size), the pre-sorted jobs of order  $j$  are assigned sequentially, each going into the bin (machine) with the largest workload in which it still fits. If all the jobs can be assigned to the machines without exceeding  $t$ , then *BFD* is considered “successful”.

If we specify for  $t$  a lower bound  $C_L(j)$ , and an upper bound  $C_U(j)$ , by trying *BFD* with different values of  $t$  in between  $C_L(j)$  and  $C_U(j)$ , schedules of different length can be generated. If the processing times are integers, a binary search procedure would make the algorithm run faster. Using a binary search procedure, we can initially try *BFD* with  $t = (C_U(j) + C_L(j))/2$ ; whenever *BFD* succeeds, we let  $C_U(j) = t$ ; otherwise, let  $C_L(j) = t$ . This procedure is repeated until  $t$  cannot be updated any more, or until after a specified number of iterations, say  $\mathcal{I}$ . The schedule obtained by trying *BFD* with the latest  $C_U(j)$  as  $t$  is chosen.

Now we fix an initial setting for  $C_L(j)$  and  $C_U(j)$ . Before the jobs of order  $j$  are assigned, we denote the smallest workload of the  $m$  machines as  $C_{min}$ . It is easy to see that, an initial lower bound  $C_L(j)$  can be set as  $C_L(j) = C_{min} + \max\{\sum_{l=1}^k p_{lj}/m, p_{1j}, \dots, p_{kj}\}$ , since  $C_L(j)$  is no larger than the completion time of an optimal assignment. As for the initial upper bound  $C_U(j)$ , we can set it as the completion time of order  $j$  obtained by a trial assignment using the *BIN* rule. In case the *BFD* would not be successful even with the initial upper bound as its target completion time, we accept the assignment by the *BIN* rule.

The various ways of combining sequencing rules with assignment rules lead to twelve different heuristics. However, we focus only on those that appear the most promising: Four heuristics based on *WSTP*, namely, *WSTP-LS*, *WSTP-LPT*, *WSTP-BIN*, *WSTP-MF*; One heuristic based on *WSLM*, namely, *WSLM-LPT*; One heuristic based on *WSMM*, namely, *WSMM-MF*. By simple analysis, the time complexity of each heuristic is as follows:

$$\begin{array}{ll}
\textit{WSTP-LS}: O(kn \lg m + n \lg n); & \textit{WSTP-LPT}: O(kn \lg km + n \lg n) \\
\textit{WSTP-BIN}: O(kn \lg k + n \lg n + knm); & \textit{WSTP-MF}: O(kn \lg k + n \lg n + \mathcal{I} knm) \\
\textit{WSLM-LPT}: O(kn \lg km + n \lg n); & \textit{WSMM-MF}: O(kn \lg k + n \lg n + \mathcal{I} knm)
\end{array}$$

### 3.2 Dynamic two-phase heuristics

The second class of heuristics are referred to as *dynamic two-phase heuristics*. In these heuristics, the sequence of the orders is not fixed prior to the assignment of the various product types to the machines, i.e., the sequence is determined dynamically. The heuristics use the *LPT* rule, the *BIN*

rule or the *MF* rule to assign the jobs to the machines. However, to determine the next order to be sequenced, a greedy approach is applied to make a trial assignment of the product types of all remaining orders by using one of three rules, and the next selected order  $j^*$  satisfies

$$j^* = \arg \min_{j \in \Omega} \left\{ \frac{C_j - C_{j'}}{w_j} \right\},$$

where  $\Omega$  is the set of unscheduled orders, and  $C_{j'}$  is the finish time of the order that was scheduled immediately before order  $j^*$ . Ties may be broken arbitrarily. In case  $C_{j^*} < C_{j'}$ , we can shift forward all those jobs of  $j^*$  assigned to each machine, and put them before all jobs of  $j'$  on that machine. Now after this shift operation, if there exists another order  $j''$  such that  $C_{j^*} < C_{j''}$ , we carry out the same shift operation between  $j^*$  and  $j''$ . This procedure is repeated until such a case does not occur any more. Clearly, with such a shift operation, the finish time of an order such as  $j'$  remains unchanged, whereas the finish time of  $j^*$  decreases. This post-processing procedure helps to reduce the objective cost of the schedule. The three heuristics are referred to as *weighted earliest completion time by LPT* (*WECT-LPT*), *weighted earliest completion time by BIN* (*WECT-BIN*) and *weighted earliest completion time by MF* (*WECT-MF*), respectively. The first two heuristics are generalizations of the unweighted versions in Blocher and Chhajed (1996), while *WECT-MF* is new. Natural implementation of each heuristic requires  $n^2$  runs of the respective assignment rule. Thus, the running times of these three algorithms are  $O(kn^2 \lg km)$ ,  $O(kn^2 \lg k + kn^2 m)$  and  $O(kn^2 \lg k + \mathcal{I}kn^2 m)$ , respectively.

### 3.3 Worst-Case Analyses of the Heuristics

In what follows, we analyze the performance bounds of the above heuristics. We noticed that Blocher and Chhajed (1996) had an empirical study for the unweighted version of *WSTP-LPT*, *WSTP-BIN*, *WECT-LPT*, and *WECT-BIN*. Yang and Posner (2005) had a worst-case analysis for the unweighted version of *WSTP-LPT*, and showed it has a tight bound of 6/5 for the unweighted problem with only two machines.

When  $n = 1$ , it is clear that the problem becomes  $P \parallel C_{max}$ . The results for  $P \parallel C_{max}$  imply the following:

**LEMMA 6** *For  $PF \mid \Pi k \mid \sum w_j C_j$ , the worst-case ratio of algorithms that use LS and LPT cannot be less than  $2 - \frac{1}{m}$  and  $\frac{4}{3} - \frac{1}{3m}$ , respectively.*

We can show the following result for the four heuristics based on the *WSTP* sequencing rule:

**THEOREM 7** *For  $PF \mid \Pi k \mid \sum w_j C_j$ , each of *WSTP-LS*, *WSTP-LPT*, *WSTP-BIN*, and *WSTP-MF* has a worst-case bound of  $2 - \frac{1}{m}$ .*

It turns out that the remaining five algorithms, namely, *WSLM-LPT*, *WSMM-MF*, *WECT-LPT*, *WECT-BIN* and *WECT-MF*, can perform very badly. To see this, consider the following example:

- Let  $w_j = 1$ ,  $j = 1, 2, \dots, n$ .
- Let  $x = \rho \cdot m$ , where  $0 < \rho < 1$ ; let  $\epsilon > 0$ .
- Each order  $j = 1, 2, \dots, x$  requests  $m$  product types, each of which requires 1 unit of processing.
- Each order  $j = x + 1, x + 2, \dots, x + m(m - x)$  requests only 1 product type requiring  $1 + \epsilon$  units of processing.

It is easy to see that each heuristic

$$H \in \{WSLM-LPT, WSMM-MF, WECT-LPT, WECT-BIN, WECT-MF\}$$

produces the same schedule, with the objective value being

$$\sum w_j C_j(H) = \sum_{j=1}^x j + m \cdot \sum_{j=1}^{m-x} (x + j \cdot (1 + \epsilon)).$$

On the other hand, it is also easy to see that an optimal schedule has an objective value of

$$\sum w_j C_j(OPT) = m \cdot \sum_{j=1}^{m-x} j \cdot (1 + \epsilon) + \sum_{j=1}^x (j + (m - x)(1 + \epsilon)).$$

It can be determined that

$$\lim_{m \rightarrow \infty} \left( \lim_{\epsilon \rightarrow 0} \frac{\sum w_j C_j(H)}{\sum w_j C_j(OPT)} \right) = \frac{1 + \rho}{1 - \rho}.$$

Thus, when  $\rho$  is close to 1, the above ratio can be arbitrarily large. This implies that the performance ratio of these heuristics is not bounded by any constant. Actually, we can show that the upper bound is  $m$ .

**THEOREM 8** *For  $PF \mid \Pi k \mid \sum w_j C_j$ , each of WSLM-LPT, WSMM-MF, WECT-LP, WECT-BIN, and WECT-MF has a worst-case bound of  $m$ .*

However, it is not clear whether this bound is tight or not. Note that the above example is not a tight example.

### 3.4 Empirical Analysis of the Heuristics

We generate problem instances with different sizes that are determined by  $n$ ,  $m$  and  $k$ , where  $n \in \{20, 50, 100, 200\}$ ,  $m \in \{2, 5, 10, 20\}$  and  $k \in \{2, 5, 10, 20, 50, 100\}$ . For each combination of  $n$ ,  $m$  and  $k$ , 10 problem instances are randomly generated. These 10 problem instances have a similar structure and are treated as a group. To produce an instance for a combination of  $n$ ,  $m$  and  $k$ ,  $n$  orders are generated. For each order  $j$ , the number of product types  $k_j$  is generated from the uniform distribution  $[1, k]$ . Then, for each product type  $l = 1, 2, \dots, k_j$ , an integer processing time  $p_{lj}$  is generated from the uniform distribution  $[1, 100]$ . In addition, a weight for order  $j$  is randomly generated from the uniform distribution  $[1, 10]$ . In total, 960 instances are generated.

The algorithms are implemented in C++. The running environment is based on the Windows 2000 operating system; the PC used was a notebook computer (Pentium III 900Mhz plus 384MB RAM). In what follows, we study the performance of the algorithms in terms of three aspects: the frequencies at which they are the best, comparison of their average costs and comparison of their average running times. Again, due to space constraints, we omit the tables for experimental results, which are available at <http://web.njit.edu/~leung/pf-sumwjcj/tables.pdf>.

We first compare the heuristics in terms of the frequencies at which they are the best. The results show that *WSTP-MF* is the best. In addition, the tendency is that a larger  $k$  leads to a higher percentage of *WSTP-MF* that are the best. The same tendency also applies to the other two heuristics based on the *MF* assignment rule, i.e., *WSMM-MF* and *WECT-MF*, even though they are inferior to *WSTP-MF*. Thus, it would be advantageous to apply *MF*-based heuristics to instances in which each order requests many different product types. When  $k$  is small, *WSMM-BIN* is comparable to *WSMM-MF*. This enforces our observation that *MF* does not have much of an advantage when each order has only a small number of jobs. Even though the analysis in the previous section showed that the worst-case performance bounds of the heuristics based on *WSTP* are much better than the other heuristics, we could not find a clear indication from the experimental results. Thus, we need to compare the heuristics in terms of average costs in percentage. From the average costs, we have the following findings:

- *WSTP-MF* performs the best, *WSTP-BIN* performs second best.
- The four heuristics based on *WSTP* are better than the others. This agrees with the worst-case bounds obtained previously.
- When  $k$  and  $n$  are fixed, the performances of the heuristics worsen as  $m$  increases. This is also consistent with the corresponding worst-case bounds obtained previously.
- When  $n$  and  $m$  are fixed, the performances of the heuristics improve as  $k$  increases. In addition, as  $k$  increases, the differences in the performances of the various heuristics become smaller.
- When  $m$  and  $k$  are fixed, the differences between the *WSTP*-based heuristics become smaller as  $n$  increases. On the other hand, as  $k$  decreases, the differences between those heuristics which are not based on *WSTP* and the *WSTP*-based heuristics become larger.
- The performances of the heuristics are sensitive to  $k/m$ . When  $k/m$  is small, the gaps between the heuristics that are not based on the *WSTP* rule and the *WSTP*-based heuristics are large. When  $k/m$  is large, these gaps are small.

As for the speeds of the heuristics, the results show that the *WSTP*-based heuristics run faster than other heuristics.

From all of the above observations, we would recommend to use *WSTP-MF* or *WSTP-BIN* in practice.

## 4. Concluding Remarks

In this paper, we have focused on the design and analysis of approximation algorithms for two extreme cases of the customer order scheduling problem, with the minimization of the total weighted completion time as objective.

For the dedicated case, the algorithms include several priority rules as well as two LP-based algorithms. Although priority rules are easy to implement, our analysis showed that their performance guarantees vary according to the distribution properties of the processing times. By contrast, various linear programming relaxations uniformly provide tight lower bounds for approximation algorithms. However, different relaxations may result in approximation algorithm with very different performance guarantees. Fortunately, both LP-based algorithms we presented in this paper have a fixed ratio performance guarantee. According to our empirical analysis, we would recommend the use of *WECT* and  $H_{LP2}$  with  $\lambda = \sqrt{2}$ . However, with regard to solution quality, speed, memory space, and implementation complexity, *WECT* is more preferable.

For the flexible case, the heuristics considered fall in two categories, namely the sequential two-phase heuristics and the dynamic two-phase heuristics. We performed a worst case as well as an empirical analysis of nine heuristics. The analyses reveal that the four *WSTP*-based heuristics perform better than the five other heuristics, despite the fact that the four *WSTP*-based heuristics are static whereas three of the other heuristics are dynamic. This may, at first sight, appear to be an anomaly, since observations in classical scheduling problems indicate that dynamic heuristics usually perform better than static heuristics. One possible reason why the static *WSTP*-based heuristics perform better than the dynamic heuristics may be based on the fact that the dynamic selection criteria may put some orders with many jobs ahead of a large number of orders with few jobs; the cumulative cost of these orders with few jobs becomes very large, just as in the example we presented in order to show that the static *WSTP*-based heuristics are better than the dynamic heuristics. Our worst-case and empirical analyses also validate this observation. With regard to solution quality and speed, we recommend the *WSTP*-based heuristics.

## References

- Blocher, J. and Chhajer, D. (1996). The customer order lead-time problem on parallel machines. *Naval Research Logistics*, 43: 629–654.
- Coffman, E. G., Garey, M. R., and Johnson, D. S. (1978). An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1): 1–17.
- Hall, L. A., Schulz, A. S., Shmoys, D. B., and Wein, J. (1997). Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22: 513–544.
- Leung, J. Y.-T., Li, H., and Pinedo, M. (2005a). Order scheduling models: An overview. In G.Kendall, E.K.Burke, S.Petrovic, M.Gendreau (eds.), *Multidisciplinary Scheduling: Theory and Applications*, Springer.
- Leung, J. Y.-T., Li, H., and Pinedo, M. (2005b). Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling*, to appear.
- Makhorin, A. (2004). GNU linear programming kit: Reference manual (Version 4.4).
- Queranne, M. (1993). Structure of a simple scheduling polyhedron. *Mathematical Programming* 58: 263–285.
- Sung, C. and Yoon, S. (1998). Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. *International Journal of Production Economics*, 54: 247–255.
- Wang, G. and Cheng, T. (2003). Customer order scheduling to minimize total weighted completion time. *Proceedings of the 1st Multidisciplinary Conference on Scheduling: Theory and Applications*, pages 409–416.
- Yang, J. and Posner, M. E. (2005). Scheduling parallel machines for the customer order problem. *Journal of Scheduling*, 8: 49–74.

# **Retractable Contract Network for Distributed Scheduling**

**E.P.K. Tsang<sup>1</sup>, T.Gosling<sup>1</sup>, B. Virginas<sup>2</sup>, C. Voudouris<sup>2</sup> & G. Owusu<sup>2</sup>**  
**Submission to MISTA 2005**

**<sup>1</sup>Department of Computer Science, University of Essex**

**<sup>2</sup>BT Research Laboratories**

## **Abstract**

This paper is about distributed scheduling where individual agents have potentially conflicting interests. It is motivated by BT's workforce scheduling problem, where multiple service providers have to serve multiple service buyers. The service providers and buyers all attempt to maximize their own utility. The overall problem is a multi-objective optimization problem; for example, one has to maximize completion rates and service quality and minimize travelling distances. In this paper, BT's problem is modelled as an open constraint optimization system.

Standard contract net is a practical strategy in distributed scheduling where agents may have conflicting objectives. In this paper, we have introduced a retractable contract net protocol, which we call RECONNET, that supports hill-climbing in the space of solutions. It is built upon a job-release and compensation mechanism. RECONNET is a general protocol, which could be used to implement complex meta-heuristic algorithms such as Tabu Search and Guided Local Search.

A system based on RECONNET has been implemented for BT's workforce scheduling problem. The software, which we call ASMCR, allows the management to have full control over the company's multi-objectives. The manager generates a Pareto set of solution by defining, for each buyer and seller, the weights given to each objective. ASMCR gives service buyers and sellers ownership of their problem and freedom to maximize their performance under the criteria defined by the management. ASMCR took 5 to 15 minutes to complete when tested on real-sized problems. It has potential to be developed into practical solutions to BT's workforce scheduling problem.

**Keywords:** scheduling, constraint optimization, multi-objective optimization, meta-heuristic search

# 1. Introduction

## 1.1. Distributed scheduling, motivations

In many real life scheduling problems where multiple entities have to work together, centralised scheduling is very difficult. For example:

1. This will be the case when different entities have potentially conflicting interest. Should scheduling be done centrally, independent entities may not inform the central scheduler all the relevant information (game playing).
2. It will also be the case when each entity has many constraints to consider, and these constraints change dynamically. For example, jobs or staff availability may change frequently. In such situations, it may be too tedious to inform a central scheduler all the minor considerations and changes.
3. Human factors may lead to local considerations not being conveyed to the central scheduler. For example, a local manager may take staff emotion into consideration and apply discretion in scheduling his staff.

In such cases, distributed scheduling may be more productive. This paper uses a case study to present a general strategy in distributed scheduling.

Distributed constraint satisfaction is a well studied topic which is highly relevant to distributed scheduling. Most of the work in distributed constraint satisfaction involves cooperative agents, where agents work together to achieve some common goals; for example see [2][10][14][18][28][29][30]. In this paper, motivated by real life applications, we study distributed constraint satisfaction problems where agents may have conflicting goals. Work in this area is relatively scarce, but see [4].

## 1.2. BT's distributed workforce scheduling problem

This research is motivated by BT's workforce scheduling problem, which involves multiple agents with potentially conflicting objectives. In this paper, the problem is formulated as a distributed scheduling problem, which is described below and formalized in the next section [21][22].

BT's workforce is organized in regions. Each region has a Service Buyer (or *Job Agent*) and a Service Seller (or *Engineer Agent*). These agents are managed by a central manager. Each service buyer has a set of jobs to be completed. Each service seller manages a set of engineers, who can be assigned to jobs. The service buyers buy services from the sellers by inviting them to bid for individual jobs. The service sellers base their bidding on which engineers could service which job (depending on availability and skills), the engineers' preferences and their travelling distances to the jobs. The buyer selects bids based on preferences and travelling distances (information supplied by the sellers). Offers are made to selected bids. These offers are binding, which means the buyer will have to honour a contract as soon as it is accepted by the seller. The sellers may receive multiple offers that require the same engineer. In this case, it uses its utility criteria to select offers. The question is how to design a protocol to enable efficient assignments of engineers to jobs.

The buyers and sellers must maximize their utility. The manager is responsible for maximizing the company's utility. The manager is able to achieve this by defining the utility function of each buyer and seller. The manager's problem is a multi-objective optimization problem: it has to strike a balance

between job completion rates, service quality, travelling distances and other objectives, which will be elaborated later.

## 2. Modelling BT's distributed workforce scheduling problem

### 2.1. Overview of the problem

The following is a specification of the components in BT's workforce scheduling problem. The challenges are also highlighted:

#### 1. The Manager

As mentioned above, the manager's problem is a multi-objective optimization problem. Its role is to build a Pareto set of schedules for the end-user to choose from. This is achieved by passing to each buyer and seller an amalgamation function that defines the weights of each agent. The agents may be given different amalgamation functions.

The manager's challenge is in how to define the amalgamation functions to pass to the buyers and sellers, in order to represent the company's interest. In multiple runs, it attempts to build a "good" Pareto set of schedules. The quality of a Pareto set can be defined in many ways (see, for example, [13] for a good survey). In this project, the control of what Pareto set to produce is left entirely to the user.

#### 2. Service Buyers

Each Buyer handles a set of jobs that need to be done. It has to find a contract for each job, by communicating with service sellers. The Buyer's job is to formulate invitations to Service Sellers to bids for jobs. The buyer may (a) invite bids for one job at a time, or (b) invite bids for more than one job at a time. In the latter case, jobs can be bundled<sup>1</sup>. When bids are received, the Buyer is responsible for allocating jobs to the appropriate Seller. The Buyer then makes offers for those bids (binding) but these might not be accepted by the Seller.

If needed, it can also contact other buyers for *contract-release*, which means offering other buyers as well as the seller compensation to give up their contracts (this will be formally defined later).

The buyer has multi-objectives, but at any point, it is given an amalgamation function by the manager. So it is effectively dealing with a single objective optimization problem. A buyer would attempt to find good solutions to its problem through a local search, which is facilitated by the contract-release protocol.

The buyer's challenge is in how to hill-climb to improve its utility according to the amalgamation function given by the manager.

#### 3. Service Sellers

Each seller handles a number of engineers. Engineers may have different abilities and constraints. The Seller's job is to bid for jobs. A bid can logically be described as a pair (Pref, Dist), where

---

<sup>1</sup> For any job that takes more than  $x$  days to finish and is more than  $y$  miles from the engineer, accommodation allowance is given. It is possible that Sellers may want to bundle jobs that are located in close proximity geographically.



Pref is the preference and Dist is the distance. It may return just one, or the set of all available bids, or a subset (possibly Pareto set) of (non-dominating) bids for each job. Note that there are typically more jobs than those that the engineers can handle. Sellers can offer partial solutions to requests, e.g. “we cannot do all 3 bundled jobs in those 5 days, but we can do the first 2 jobs in the first 3 days”. In this paper, we assume that the sellers are relatively passive: when a seller receives invitations to bid for jobs, it is required to make all the non-dominated bids (as in multi-objective optimization). Then it waits for (binding) offers from the buyers, to which it may accept (binding) or decline. Each seller has multiple objectives, but, like the buyers, at any time, it is given an amalgamation function by the manager on the weights of each objective. So effectively, a seller deals with a single objective optimization problem.

The seller’s challenge is in how to schedule its engineers effectively, though only a simple strategy will be considered in this project.

Naturally, most of the jobs are serviced by the seller in the same region as the buyer. However, better schedules could result in cross-regional services. The challenge in this work is to define a mechanism for generating good solutions to BT’s dynamic workforce scheduling problem. Special attention is paid to the communication protocol among the agents. We want a protocol that supports the implementation of advanced heuristic methods, such as Tabu Search [6][7][8][9] and Guided Local Search [11][12][23][24][27].

## 2.2. A formal definition of the business model

In this section, we provide a rigorous description of the business model.

### ***Job information:***

Each job is described by a tuple:

$$(Loc, Sk, D, SD, P)$$

where Loc is a location, which describes the coordinates (x, y) of the job. Sk is the skill required, which ranges from 1 to 9 (which is not a rank order). D is the duration of the job, in terms of number of days. SD is the starting day, which could be 1 to MaxD, where MaxD is the number of days that we plan ahead (MaxD was set to 7 in our experiments, i.e. to plan 7 days ahead). P is a price, which relates to the value or importance of the job (this may be determined by the manager).

### ***Engineer information:***

Each engineer is described by:

$$(Loc, LSP, SA, OT)$$

where Loc is a location, which describes the coordinates (x, y) of the engineer. LSP is a list of (*skill, preference*) pairs. Each pair describes the preference of this engineer on jobs that require the specific skill. Preferences range between 1 and 9, with 1 meaning most favourable and 9 meaning least favourable. SA is the *surplus availability* of the engineer, which is a list of the days on which this engineer is available. OT is a Boolean variable on overtime availability (0 means no overtime is allowed by this engineer, 1 means overtime allowed).

The problem of each buyer and seller was formulated as an *open constraint optimization system* [20]. An open constraint optimization system is a constraint satisfaction problem [19] where some constraints are not entirely within the control of the problem solver itself. It is, instead, shared with other solvers. This means to check whether such constraints are satisfied, a problem solver has to negotiate with other

solvers. It is therefore an open system, as opposed to a closed one, where all constraints are within the solver's control. (The readers should not confuse this with the Open Constraint Satisfaction Problem defined by Faltings & Macho-Gonzalez [3], where the number of variables may change dynamically.)

### The Buyer's Model:

The problem of buyer  $b$  can be formulated as an open constraint satisfaction model:

$$(Z_b, D_b, C_b, E_b, f_b, Ag_b, EtA_b, CP_b)$$

where

$Z_b = \{s[1], s[2], \dots, s[n_b], p[1], p[2], \dots, p[n_b], d[1], d[2], \dots, d[n_b]\}$ , where  $n_b$  is the number of jobs that  $b$  has,  $s[i]$  represents the service seller that  $b$  appoints to do job  $i$ ,  $p[i]$  represents the preference and  $d[i]$  represents the distance for serving job  $i$ , which are proposed by the service sellers and accepted by the buyer;

$Ag_b$  is the set of seller agents who  $b$  has contact to;

$D_b$  is a function that defines the domain of the variables in  $Z_b$ , as in constraint satisfaction [19]. For all  $i$ ,  $D_b(s[i]) = Ag_b$  plus  $\phi$ , which means  $s[i]$  could be assigned one of the seller agents, or assigned no seller at all (which is represented by the value  $\phi$ );  $D_b(p[i]) = [0..9]$ , which means  $p[i]$  could be assigned a value 0 to 9, with 0 meaning the job is not served, 1 to 9 are preferences in the service. For all distance variables  $d[i]$ ,  $D_b(d[i]) = \mathbf{R}$ ;

$C_b$  represents a set of internal constraints, which is an empty set in this case, i.e. there are no constraints on what value  $b$  assigns to the variables;

$E_b = \{ E_b(s[i], p[i], d[i]) \mid i = 1.. n_b \}$ , where  $E_b(s[i], p[i], d[i])$  is a constraint on the values of  $s[i]$ ,  $p[i]$  and  $d[i]$ , restricting the values that they can take simultaneously; the values of  $p[i]$  and  $d[i]$  are to be determined by external agents,  $s[i]$  indicates the seller that  $b$  assigns job  $i$  to; it is assigned by  $b$ , depending on the bids by the sellers;

$f_b$  is the objective function for  $b$ . It is a multi-objective function;

$$f_b = (reve_b, failure_b, pref_b, dist_b, comm_b)$$

- Maximize  $reve_b = (\sum_{i, p[i]>0} \text{Price}[i])$ , which is to maximize total revenue, where  $\text{Price}[i]$  is a constant;
- Minimize  $failure_b = (\sum_{i, p[i]=0} 1)$ , which is to minimize the number of unassigned jobs;
- Minimize  $pref_b = (\sum_{i, p[i]>0} p[i])$ , which is to minimize the total preference;
- Minimize  $dist_b = (\sum_{i, p[i]>0} d[i])$ , which is to minimize total distance travelled;
- Minimize  $comm_b = \text{number of communications}$ .

The buyer  $b$  attempts to maximize its utility:

$$\text{Utility} = k_1 \times reve_b - k_2 \times failure_b - k_3 \times pref_b - k_4 \times dist_b - k_5 \times comm_b + \text{Trade}$$

where the weights  $k_1$  to  $k_5$  are given by the manager; Trade is income from other buyers – payment to other buyers for contract-release;

$EtA_b$  is the mapping from each external constraint  $E_b$  to a seller agent; i.e.  $EtA_b(E_b(s[i], p[i], d[i]))$  returns a value in  $Ag_b$ ; this indicates that the values of  $p[i]$  and  $d[i]$  are to be determined by all seller agents ( $Ag_b$ ) in communication with  $b$ ;

$CP_b$  is the communication protocol. Here we assume the following protocol:

1. The buyer  $b$  sends a set of invitation to bid to seller  $s$ ; each invitation is

$$(\text{Job\_ID}, \text{Job\_information})$$

where Job\_ID is the identity of the job, and Job\_information is a tuple as defined above:

$$(\text{Location}, \text{Min\_Skill}, \text{Duration}, \text{StartDay}, \text{Price})$$

2. The seller  $s$  sends a set of pairs of values to  $b$  for instantiating  $(p[i], d[i])$
3. The buyer  $b$  offers  $s$  a contract, which comprises a pair of  $p$  and  $d$  values
4. The seller  $s$  accepts the contract (and commits its resources) or declines the offer, in which case, go back to Step 3 (where  $b$  could offer a contract to another seller)

Once variables  $p$  (preferences) and  $d$  (distances) are communicated between  $b$  and  $s$ , they can be seen as *contracts* between the buyer and the seller.

### The Seller's Model:

The problem of seller  $s$  can be formulated as a dynamic open constraint satisfaction model<sup>2</sup>:

$$(Z_s, D_s, C_s, E_s, f_s, Ag_s, EtA_s, CP_s)$$

where

$Z_s = \{e[1], e[2], \dots, e[N], p[1], p[2], \dots, p[N], d[1], d[2], \dots, d[N]\}$ , where  $N$  is the total number of jobs that  $s$  has been invited to bid for and  $s$  is still in contention (i.e. the buyer has not yet assigned the job to another seller);  $e[i]$  represents the engineer that is assigned to do job  $i$ ;  $p$  and  $d$  represents preferences and distances as defined in buyers;

$Ag_s$  is the set of buyer agents who  $s$  has contact to;

$D_s$  is a function that defines the domain of the variables in  $Z_s$ , as in constraint satisfaction [19]. For all  $i$ ,  $D_s(e[i]) =$  the set of engineers plus  $\phi$ , which means  $e[i]$  could be assigned one of the engineers, or assigned no engineer at all (which is represented by  $\phi$ );  $D_s(p[i]) = [0..9]$ , which means  $p[i]$  could be assigned a value 0 to 9, with 0 meaning the job is not served, 1 to 9 are preferences in the service; For all distance variables  $d[i]$ ,  $D_s(d[i]) = \mathbf{R}$ ; default values for  $p$  and  $d$  are 0, which means no engineer is assigned to job  $i$  until commitment is made (by  $s$ );

$C_s$  represents the internal constraints that governing the feasibility of the engineers doing the jobs; this involves the availability and skills of the technicians;

$E_s = \{E_s(e[i], p[i], d[i]) \mid i = 1..N\}$ , where  $E_s(e[i], p[i], d[i])$  is a constraint on the values of  $e[i]$ ,  $p[i]$  and  $d[i]$ , restricting the values that they can take simultaneously; the values of  $e[i]$ ,  $p[i]$  and  $d[i]$  are proposed by  $s$ , to be approved by the buyer;

$f_s$  is the objective function for  $s$ . Associated to each job  $i$ ,  $Price([i])$  is a constant given by the Buyer.  $f_s$  a multi-objective function:

- Maximize jobs done  $JD = (\sum_{i, p[i]>0} 1)$
- Minimize distance travelled  $DT = (\sum_{i, p[i]>0} d[i])$
- Maximize load balancing  $LB = -(\sum_{i=1,N} (free\_time[i] - Mean\_free\_time)^2)/N$  where  $N$  is the number of engineers
- Minimize redundancy  $RD = (\sum_{i=1,N} (free\_time[i]/time\_available[i]))/N$

The seller  $s$  attempts to maximize its utility:

$$Utility = k_1 \times JD - k_2 \times (DT)^2 + k_3 \times LB - k_4 \times (RD) + CR$$

Where the weights  $k_1$  to  $k_5$  are given by the manager; Trade is the income from buyers for contract-release.

$EtA_s$  is the mapping from each external constraint  $E_s$  to a buyer; i.e.  $EtA_s(E_s(e[i], p[i], d[i]))$  equals the service buyer for job  $i$

<sup>2</sup> The seller's problem does not have to be formulated as an open constraint satisfaction system. It can be formulated independent of the buyer's model. Here it is formulated as a dynamic scheduling model, with new variables being created when the seller receives requests from the buyer, and variables removed when the buyer does not select this seller.

CP<sub>s</sub>: see E<sub>b</sub> in the Buyer's model

### **The Manager (Centralized Coordination):**

The objective is to balance failure of all domains. In this research, the following objectives are considered:

- Minimize the average failure rate:  $(\sum_{b \in \text{Buyer}} FR_b) / NB$   
where  $FR_b = \text{failure}_b / n_b$ , where  $\text{failure}_b$  is the number of failures for buyer  $b$  and  $n_b$  is the number of jobs that  $b$  has, NB is the number of buyers;
- Minimize the total distance travelled in all jobs done;
- Minimize the average preferences of all jobs done;
- Maximize failure-imbalance =  $(\sum_{i=1, Nb} (\text{failure}_i - \text{Mean\_failures})^2) / Nb$   
where Nb = number of buyers in the system.

The Manager's overall objective is a multi-objective function. It is not necessary to combine the above measures – this is a job left to the end-user. However, the manager should attempt to produce a Pareto set of solutions. This can be done by giving the buyer/seller agents different sets of weights for the different buyers and seller measures (the  $k_i$ 's mentioned above). For example, the  $k_i$ 's can be used to empower individual buyers to increase their bargaining power so as to reduce their failure rates. The manager may ask the agents to:

- Schedule from scratch; or
- Improve on the previous schedule

## **3. The RECONNET protocol for BT's problem**

This work is loosely based on, and extended from the contract net protocol, a fundamental protocol for distributed artificial intelligence [1][5][16][17]. Some modifications are motivated by BT's workforce scheduling problem:

1. In BT's problem, each agent looks after its own interest. They cooperate in order to achieve their common goals (of getting the jobs done).
2. BT's problem is an optimization problem. The operation involves millions of Pounds, and therefore a small percentage in saving could mean a lot in real terms. Therefore, it is worth investing computation time to reduce cost.
3. The problem is a dynamic problem: new jobs arrive at all time, and availability of staff may change. Therefore, there is a limit in how much computation time one can spend on scheduling.
4. There are hundreds of jobs and engineers in each region. Each job could potentially be served by most engineers, though not all of them are good matches (for example, due to travelling distance). If all possible contracts were communicated, the amount of communication would be impractically large. Therefore, we must attempt to reduce communication.

Given points 2 and 3 above, local search would be a good candidate. A retractable contract net protocol, which we call RECONNET (which stands for Retractable Contract Net), that facilitates hill-climbing is designed. Used by the manager in iterations, RECONNET will enable us to implement any meta-heuristic methods such as Tabu Search and Guided Local Search.

### **3.1. Building contracts**

A *contract* is a tuple:

(ID, Seller, Buyer, Preference, Distance)

where ID is the identity of the contract, Seller and Buyers are the agents involved, Distance and Preferences are as defined above.

Following is the mechanism for building initial contracts in RECONNET:

1. **Invitation to bid:** The buyers initiate contract building. The buyers would invite all sellers (or a subset of sellers, if it sees fit) to bid for all the jobs currently available.
2. **Bidding for jobs:** Given an invitation to bid for a job  $j$ , a seller will find all engineers who are available and qualified to do  $j$ . Since each qualifying engineer may have different locations and preferences for  $j$ , the seller would build up a list of (preference, distance) pairs. To reduce the volume of communication, the seller will only send to the buyer contract proposals that do not dominate each other. In other words, if engineers A and B can both serve job  $j$ , but A is closer to the location of  $j$  and has higher preference than B, then the seller will not bid a contract for B to do  $j$ .
3. **Making offers to sellers:** After receiving bids from the sellers, the buyer will build a priority queue for each job. The bids are ordered by their contributions to the buyer's weighted objective function. Jobs are then offered to the proposed contracts that are at the heads of priority queues. In a model, in every round, the buyer makes an offer on every job for which a contract has yet to be secured. This does not prevent the buyers from adopting more complex strategies. When the buyer makes an offer to a seller, the offer is binding. That means if the seller accepts the job, the buyer is not allowed to cancel it and offer the job to another seller.
4. **Acceptance of offers:** In this project, we assume that the seller will accept an offer as long as an engineer is available. When a seller receives multiple contracts that require the same engineer at the same time, it will pick the offer that maximizes its objective function. This simple strategy can be replaced by a more sophisticated algorithm in the future (for example constraint satisfaction techniques and iOpts [25][26]), in order to maximize the chance of doing more jobs (based on probabilities and past experience). For example, a seller may consider fairness amongst the engineers.

When new jobs arrive dynamically, Steps 1 and 2 are invoked. Contracts are being built through repeated steps 3 and 4. At any time, the buyer maintains the status of all the jobs. For example, Table 1 shows a snap shot of the status of four jobs.

**Table 1. Sample priority queue for a particular service buyer**

|                                                       | Bid 1 | Bid 2 | Bid 3 | Bid 4 | Bid 5 | Bid 6 |
|-------------------------------------------------------|-------|-------|-------|-------|-------|-------|
| Job1                                                  | ×     | ×     |       |       |       |       |
| Job2                                                  | ✓     |       |       |       |       |       |
| Job3                                                  | ×     | ×     | ×     | ×     |       |       |
| Job4                                                  | ×     | ×     | ✓     |       |       |       |
| Keys: ✓ means contract secured, × means bids declined |       |       |       |       |       |       |

Table 1 shows that two bids have been received for Job 1. The offers to both bids have been declined. Contracts have been secured for both jobs 2 and 4. A contract has yet to be secured for job 3, which has one bid left to be unexplored. At this point, the buyer may invite further bids for job 1. The sellers may bid contracts that were dominated by Bids 1 and 2, or contracts that become available due to change of engineers' availability.

### 3.2. Contract-release

To facilitate hill-climbing, a *contract-release* mechanism is introduced. This is a mechanism for one service buyer to release its contract to another buyer, to enable to latter to complete jobs with higher

utility (according to the manager's amalgamation functions). Suppose buyer  $s$  has a job  $j$  that cannot be served by any seller given their current commitments. With information provided by the sellers,  $s$  may offer to pay another buyer to release its contract, and therefore releasing the engineers to do  $j$ .

A *contract-release* is a tuple:

$$((ID, Seller, Buyer, P, D), (CR\_ID, Owner), Cost)$$

where  $(ID, Seller, Buyer, P, D)$  is a contract that the seller offers to the buyer who invites bidding.  $CR\_ID$  is the identity of another contract, which must be released before this bid can be implemented.  $Owner$  is the owner of the contract with  $CR\_ID$ .  $Cost$  is a cost (which could be 0), from the seller's point of view, of changing the contract. A cost is required if, by changing from the released contract to the new contract, the seller's utility is reduced. For example, the new contract may require more travelling by the seller's engineers.

To complete a contract-release, the Buyer must request the *Owner* to release its contract. It will do so by offering compensation to *Owner*. The compensation is determined by the gain in completing the contract, based on the Buyer's objective function, less the compensation requested by the seller.

Before *Owner* accepts this request, it will look for an alternative contract. This may involve (a) making an offer to the next bid in the priority queue, or, in the case of the queue being reduced to empty, (b) invite further bids from the sellers. A bid from a seller could involve further contract-release. *Owner* decides to accept or reject a contract-release request with the following consideration:

- Releasing the contract for a (typically) inferior contract involves a cost to *Owner*.
- *Owner* will accept the contract-release if compensation received covers the cost of switching to the next available bid.

Since every request for contract-release requires compensation, the chain will not continue indefinitely. The contract-release mechanism is shown in Figure 1.

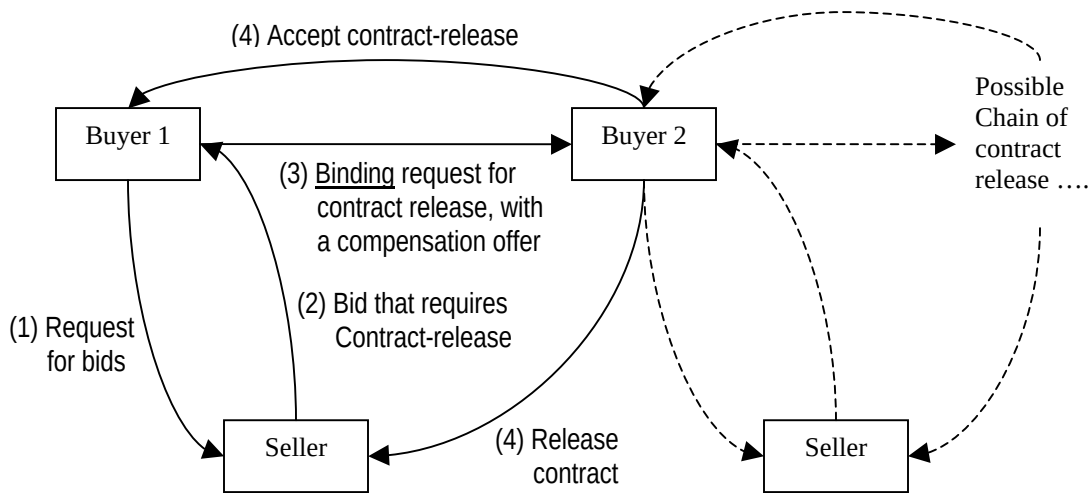


**Figure 1 The contract release mechanism**

### 3.3. Decisions in contract-release

A buyer  $s$  may receive more than one potential release contract for completing a job. Each of these release contracts is owned by a buyer. This owner could be another buyer, but it could also be  $s$  itself. Which release contract should  $s$  choose? We assume that  $s$  will always favour contract-release that involves itself. This is because releasing a contract by itself tends to be the cheapest. Besides, by knowing the availability of options in its other jobs,  $s$  has a better chance of completing the contract release chain. Consuming too many rounds of communication on one contract means having less time for other contracts (given that the whole system runs for a limited amount of rounds). Picking a contract that the buyer itself owns saves communication rounds and increases the chance of completing the chain.

If a job requires more than one day to complete, it is possible that it can only be done by releasing two or more contracts. This will require the buyer to build a tree, as opposed to building a chain of contract-releases. This becomes more complex. A buyer will have to heuristically decide how much to pay each other buyer for a contract-release. Besides, since contract-release offers are binding, the buyer will have to pay others who have released their contract even if the whole tree of release cannot be completed (i.e. this buyer pays the contract-release compensations to other buyers for no return) We have decided to allow only one contract-release per job at this stage because, from both local and global point of view, the chance of benefiting of completing one job by releasing two is not high.

## 4. ASMCR, an implementation of RECONNET

ASMCR, which stands for Automatic Synchronising Multiple Communication Rounds, is a system that is applied to BT's workforce scheduling problem. It implements the RECONNET protocol for reallocation of jobs. It is written in a way that allows easy adaptation to other problems.

ASMCR was written in Java and makes use of WebLogic, a commercial server platform that supports message passing between processes. Specifically the software uses JMS (Java Messenger Service) for communication. Note that JMS handles messages as broadcasts, but that is just an implementation issue and should not be confused with our logical decision mentioned in Section 3.1 point 1. If every seller agent sends to the buyers all the available contracts, the amount of communication would be very high. In ASMCR, the seller agents only offer non-dominated contracts, i.e. contracts that are not dominated (in Pareto sense) by other contracts by the same seller.

### 4.1. Overall Control in ASMCR

The manager's task is to find a Pareto set for the user. There are many ways to conduct the search. In this project, we assume that the Management is responsible for finding the Pareto set. This leaves the sellers and buyers standard optimization problems.

The following control strategy is used in ASMCR.

#### **Procedure Overall\_Control\_by\_Management**

*/\* the management defines and iteratively revises the cost function for job allocation \*/*

Repeat

1. Send each agent  $x$  (which is either a buyer or a seller) a cost function  $F_x$ , which determines how multi-objectives are amalgamated
2. Run **Control\_for\_Jobs\_Allocation**(  $F_x$ 's )  
*/\* agents may be asked to build on their previous assignments or schedule afresh \*/*
3. The buyers and sellers submit their final plan to the management

4. The management revises the function
- Until all sellers have received an End of Process from every buyer, or a maximum number of iterations

By providing the agents with different sets of  $F_x$ , the manager can make sure that it gets a set of different assignments to make a Pareto set of solutions. In practice, human users like to have control over the system. Therefore, they are given the authority (and responsibility) to choose the solution.

## 4.2. Job Allocation Procedure in ASMCR

Section describes how a contract is built for a job. In this section, our focus is on the agents' control strategies.

Contracts are built through a repeated loop of communications between buyers and sellers. Each round of communication is divided into four phases. The loop terminates when all buyers are satisfied with their contracts for all their jobs or the maximum number of iterations have been reached.

### **Procedure Control\_for\_Jobs\_Allocation (Fx's)**

*/\* Fx's is a set of amalgamation functions, one for each buyer and each seller. \*/*

*/\* Buyers and sellers build their contracts by allocating jobs to technicians. \*/*

*/\* All messages can be empty, with an End of Message marker. \*/*

Repeat

1. Each buyer  $b$  sends a message to each other buyer  $b1$  (in parallel)  
The message may contain:  
Request for contract release (binding)
2. Each buyer  $b$  sends a message to each seller  $s$  (in parallel)  
The message may contain:  
Request for bids  
Request for bids with possible release of contracts  
Job offers (binding)
3. Each service sellers  $s$  responses to each buyer who has sent  $s$  a message  
The message may contain:  
A list of options for each job sent by  $s$   
A list of contract-release options for each job sent by  $s$   
Accept offer  
Decline offer
4. Each buyer that receives a contract-release request replies to the requesting buyer  
The message may contain:  
Agreement to release contract (binding)  
Decline in contract release  
Wait for further decisions

Until all sellers have received an End of Process from every buyer, or the search has reached a maximum number of iterations

Communication between the agents is synchronised in ASMCR. The ordering of the four phases is subtly important. The arrangement in Procedure Control\_for\_Jobs\_Allocation allows a contract or contract-release to be completed within one round.

A technical point needs clarification here. Before buyer  $X$  accepts a contract-release from buyer  $Y$ ,  $X$  has to secure a contract to cover the job  $J$  served by the released contract. It is possible that buyer  $Y$  holds the



next best contract to cover job J. If X makes a contract-release request to Y, a cycle would be formed. Cycles are undesirable because contract-release requests are binding offers. That means, once Y has made a request to X, it has to wait for X's reply. Care has been taken in ASMCR to detect cycles in contract-release requests. We have also imposed a limit to how long a buyer will attempt to fulfil a contract-release request before it gives up.

It is worth elaborating the point that by adjusting the  $F_x$ , the Management may conduct a version of guided local search. The manager can reduce the number of unassigned jobs by modifying the cost functions given to the buyers, which could involve increasing incentives, which will increase the length of a contract revision chain.

ASMCR shares the problems with other local search methods. Being an incomplete search, optimality is not guaranteed for its solutions. It is also difficult to know when to stop. The question of when to terminate the search depends on processing power and the desirable response time for the system.

## **5. ASMCR for BT's distributed workforce scheduling problem**

To test its feasibility, ASMCR was tested on real sized data. ASMCR took 5 to 15 minutes to generate a complete plan, and conducted 50 rounds of communication. Like most hill-climbing algorithms, the more time (in terms of rounds) ASMCR is allowed to hill climb, the more chance it has in finding better or optimal solutions. In a typical run, initial solutions were generated after about 10 rounds. The rest of the rounds contributed to hill climbing.

For testing ASMCR, we generated random prices between 0.5 and 3.0 for the jobs. We used 7 buyers and 7 sellers, with 150 to 300 jobs and 150 to 300 engineers each. This is within the size range of real problems. To stress-test the algorithm, most of the experiments were generated with far more jobs than engineers.

ASMCR has been tested thoroughly using randomly generated problems. The usefulness of hill-climbing was confirmed. Figure 2 shows a scenario with three regions. Figure 2 (a) shows a situation with three jobs and two engineers. Each engineer was assigned to a job within the same region. A job in the rightmost region was not served. Suppose a new engineer in the leftmost region. When the costing is right, ASMCR was able to reschedule the assignments to serve all the three jobs, as shown in Figure 2 (b). This was only possible when contract-release was enabled.

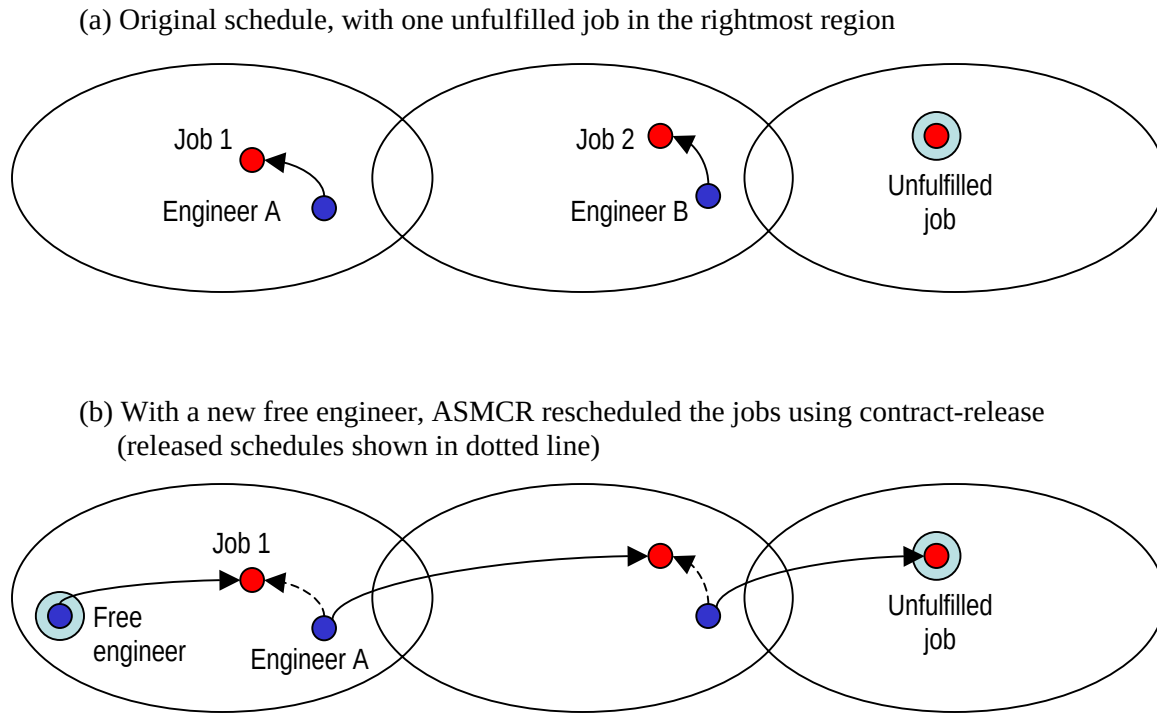


**Figure 2 The contract release mechanism enables ASMCR to revise contracts**

Experiments also confirmed that by changing the weights to the multiple objectives, the manager can reduce the number of jobs not served, travelling distance and preference (lower value in preference means better service quality).

**Table 2. ASMCR Performance**

|                                  | Base line | Change in<br>Buyer Distance | Change in<br>Buyer<br>Preference | Change in<br>Seller<br>Distance |
|----------------------------------|-----------|-----------------------------|----------------------------------|---------------------------------|
| <b>Results</b>                   |           |                             |                                  |                                 |
| Completed Jobs                   | 20.93     | 20.64                       | 20.71                            | 18.57                           |
| Contract Distance                | 35.29     | <b>34.74</b>                | 38.14                            | <b>30.92</b>                    |
| Contract Preference              | 4         | 5                           | <b>2</b>                         | 4                               |
| <b>Control Parameters</b>        |           |                             |                                  |                                 |
| Buyer Revenue ( $reve_b$ )       | 5         | 5                           | 5                                | 5                               |
| Buyer Unassigned ( $failure_b$ ) | 3         | 3                           | 3                                | 3                               |
| Buyer Distance ( $dist_b$ )      | 0.01      | <b>0.1</b>                  | 0.01                             | 0.01                            |
| Buyer Preference ( $pref_b$ )    | 0.01      | 0.01                        | <b>0.1</b>                       | 0.01                            |
| Seller Complete (JD)             | 5         | 5                           | 5                                | 5                               |
| Seller Distance (DT)             | 0.01      | 0.01                        | 0.01                             | <b>0.1</b>                      |
| Seller LoadBalancing (LB)        | 0.1       | 0.1                         | 0.1                              | 0.1                             |
| Seller Redundancy (RD)           | 0.1       | 0.1                         | 0.1                              | 0.1                             |

Table 2 provides an example ASMCR output. The figures were arrived at by averaging the results of 50 experiments for each parameter set. The configuration parameters used are shown in the lower half and directly relate to those described in Section 2.2.

Table 2 shows a baseline result with its parameters and the effect of adjusting those parameters. For instance by increasing the buyer distance penalty ( $dist_b$ ) from 0.01 to 0.1, a lower contract distance can be seen in column 3. This change has a cost: the contract preference has risen (from 4 to 5) and the number of completed jobs has been slightly reduced (from 20.93 to 20.64). Columns 4 and 5 show the effects of increasing the weights for buyer preference and sellers' traveling distance, respectively. Desired results were achieved as expected. The results show that ASMCR enables users to generate schedules of different quality by adjusting the weights of the multi-objectives.

The model described in this paper is more general than the actual set up. (That is why random problems, as opposed to real data, were used for testing.) For example, job prices have yet to be fully explored. ASMCR is a highly configurable system. This enables the system to be adapted to cope with BT's demand. Given its flexibility and efficiency, ASMCR could potentially be developed into a practical system.

## 6. Future Research

This project lays the foundations of an architecture that enables hill-climbing in building contract nets. Following are some of the directions that are worth serious research in the future (in order of importance and promise):

1. Adjustment of parameters – how the manager should adjust the parameters for amalgamation of objectives.
2. The Hill climbing procedure – how the hill-climbing protocol may be improved to maximize utilization of the resources and minimize failure.
3. Scheduling by the seller – the seller is relatively passive under the current system. This does not have to be, and in fact, should not be the case. A seller should be able to improve its performance, as defined by the objective function that it is given. For example, the current stipulation is that the seller will accept offers with maximum utility (a greedy strategy). It may benefit from delaying acceptance of offers for engineers who can serve many jobs, and committing engineers who has fewer options. This resembles the smallest-domain-first principle in constraint satisfaction [15][19].

## 7. Conclusions

This paper is about distributed scheduling where individual agents have potentially conflicting interests. It is motivated by BT's workforce scheduling problem, where multiple service providers have to serve multiple service buyers. Each agent attempts to maximize its utility. In this paper, BT's problem is modelled as an open constraint optimization system. The overall problem is a multi-objective optimization problem: one has to maximize completion rates and service quality and minimize travelling distances. The job of balancing different objectives is given to the manager. This leaves us with single-objective optimization problems to solve at the lower level.

Standard contract net is a practical strategy in distributed scheduling where agents may have conflicting objectives. In this paper, we have introduced a retractable contract net protocol that supports hill-climbing in the space of solutions. It is built upon a job-release and compensation mechanism. RECONNET is a general protocol, that could be used to implement complex meta-heuristic algorithms such as Tabu Search [6][7][8][9] and Guided Local Search [11][12][23][24][27].

A system based on RECONNET has been implemented in Java using WebLogic at BT for its workforce scheduling problem. The software, which we call ASMCR, allows the management to have full control over the company's multi-objectives. The manager generates a Pareto set of solutions by defining, for each buyer and seller, the weights given to each of their objectives. ASMCR also gives service buyers and sellers ownership of their problem and freedom to maximize their performance under the criteria defined by the management. ASMCR took 5 to 15 minutes to complete when tested on real-sized problems. It has potential to be developed into practical solutions to BT's workforce scheduling problem.

## Acknowledgements

Edward Tsang was sponsored by a BT Short Term Fellowship, 2004. Tim Gosling was sponsored by a BT-EPSRC CASE Studentship. The anonymous reviewers' comments have helped to improve this paper.

## References

- [1] Davis, R., and Smith, R.G., Negotiation as a metaphor for distributed problem solving, in: Bond, A., and Gasser, L. (ed.), Readings in distributed artificial intelligence, Morgan Kaufmann, 1988, 333-356
- [2] Durfee, E.H., Distributed problem solving and planning, in Weiss, G. (ed), Multiagent systems, a modern approach to distributed artificial intelligence, MIT Press, 1999, 121-164
- [3] Faltings, B. & Macho-Gonzalez, S., Open constraint optimization, in Rossi, F. (ed.), Proceedings, Principles and Practice of Constraint Programming (CP 2003), Lecture Notes in Computer Science 2833, Springer 2003, 303-317
- [4] Faltings, B., A budget-balanced, incentive-compatible scheme for social choice, Manuscript, Artificial Intelligence Laboratory, Swiss Federal Institute of Technology, May 2004
- [5] FIPA (Foundation for Intelligent Physical Agents), FIPA Contract Net Interaction Protocol Specification, <http://www.fipa.org/specs/fipa00029/SC00029H.html>, 2002
- [6] Glover, F., Tabu search Part I, in Operations Research Society of America (ORSA) Journal on Computing, Vol. 1, 1989, 109-206
- [7] Glover, F., Tabu search Part II, in: Operations Research Society of America (ORSA) Journal on Computing, Vol. 2, 1990, 4-32
- [8] Glover, F.W. & Laguna, M., Tabu Search, Kluwer Academic Publishers, 1997
- [9] Glover, F. & Kochenberger, G.A. (ed.), Handbook of metaheuristics, Kluwer, 2003
- [10] Luo, Q., Hendry, P. & Buchanan, I., A new algorithm for dynamic distributed constraint satisfaction problems, Research Report KEG-4-92, University of Strathclyde, 1992
- [11] Mills, P. & Tsang, E.P.K., Guided Local Search for solving SAT and Weighted MAX-SAT Problems. In Journal of Automatic Reasoning, Special Issue on Satisfiability Problems, Kluwer, Vol.24, 2000, 205-223
- [12] Mills, P., PhD Thesis, Extended Guided Local Search, Department of Computer Science, University of Essex, July 2002
- [13] Okabe, T., Evolutionary multi-objective optimization, Shaker Verlag Aachen 2004

- [14] Prosser, P., Distributed asynchronous scheduling, PhD Thesis, Department of Computer Science, University of Strathclyde, November, 1990
- [15] Smith, B.M. & Grant, S.A., Trying harder to fail first, in Prade, H. (ed.), Proceedings, European Conference on Artificial Intelligence (ECAI 98), Wiley, 1998, 249-253
- [16] Smith, R.G., The contract net protocol: high-level communication and control in a distributed problem solver, IEEE Transactions on Computers, Vol.C-29, No.12, 1980, 1104-1113
- [17] Smith, R.G., and Davis, R., Frameworks for cooperation in distributed problem solving, IEEE Transactions on Systems, Man, and Cybernetics Vol.11, No.1, 1981, 61-70
- [18] Tel, G., Distrubted control algorithms for AI, in Weiss, G. (ed), Multiagent systems, a modern approach to distributed artificial intelligence, MIT Press, 1999, 539-580
- [19] Tsang, E.P.K., Foundations of Constraint Satisfaction, Academic Press 1993
- [20] Tsang, E.P.K., Constraint satisfaction in business process modelling, Technical Report CSM-359, University of Essex, Colchester, UK, January, 2002
- [21] Ursu, M., Virginas, B. & Voudouris, C., Distributed resource allocation via local choices: general model and a basic solution, Proceedings, 8th International Conference on Knowledge-Based Intelligent Information & Engineering Systems, KES2004, Wellington, New Zealand, September 2004
- [22] Virginas, B., Ursu, M., Owusu, G. And Voudouris, C., Intelligent workforce allocation within an agent-based paradigm: central and distributed decision powers, The IASTED International Conference on Artificial Intelligence and Applications (AIA 2005), The Twenty-Third IASTED International Multi-Conference on Applied Informatics, Innsbruck, Austria, February 14-16, 2005
- [23] Voudouris, C., Guided Local Search - An Illustrative Example in Function Optimisation. In BT Technology Journal, Vol.16, No.3, July 1998, 46-50
- [24] Voudouris, C. & Tsang, E.P.K., Guided Local Search and its application to the Travelling Salesman Problem. In European Journal of Operational Research, Elsevier Science Publishers, Vol.113, Issue 2, March 1999, 469-499
- [25] Voudouris, C., Dorne, R., Lesaint, D. & Liret, A., iOpt: a software toolkit for heuristic search methods, Proc., 7th International Conference on Principles and Practice of Constraint Programming, Springer-Verlag, 2001, 716-729
- [26] Voudouris, C., and Dorne, R., Integrating Heuristic search and One-Way Constraints in the iOpt Toolkit, in Voss, S. and Woodruff, D. (ed.), Optimization Software Class Libraries, Chapter 6, pp 177-192, Kluwer Academic Publishers, 2002.
- [27] Voudouris, C. & Tsang, E.P.K., Guided local search, Chapter 7, in Glover, F. (ed.), Handbook of metaheuristics, Kluwer, 2003, 185-218
- [28] Yokoo, M., Ishida, T. & Kuwabara, K., Distributed constraint satisfaction for DAI problems, AAAI Spring Symposium on Constraint-based Reasoning, March, 1991, 191-199
- [29] Yokoo, M., Weak-commitment search for solving constraint satisfaction problems, Proc., 12th National Conference for Artificial Intelligence (AAAI), 1994, 313-318
- [30] Yokoo, M. & Ishida, T., Search algorithms for agents, in Weiss, G. (ed), Multiagent systems, a modern approach to distributed artificial intelligence, MIT Press, 1999, 165-199

# SCHEDULING ALGORITHMS FOR MASTER-SLAVE SYSTEMS\*

Joseph Y-T. Leung and Hairong Zhao

*Department of Computer Science  
New Jersey Institute of Technology  
Newark, NJ 07102, USA*

**Abstract** We consider scheduling problems in the master-slave model. In this model each job has to be processed sequentially in three stages. In the first stage, a preprocessing task runs on a master machine; in the second stage, a slave task runs on a dedicated slave machine; in the last stage, a postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. It has been shown that the problem of minimizing makespan or total completion time is NP-hard in the strong sense even if preemption is allowed. In this paper we design efficient approximation algorithms to minimize the total completion time in various settings. These are the first general results for total completion time problem in the master-slave model. We also show that these algorithms generate schedules with small makespan as well.

**Keywords:** Master-slave, scheduling, NP-hard, approximation algorithms, makespan, total completion time

## 1. Introduction

We consider scheduling problems in the master-slave model which was recently introduced by Sahni [13]. In this model, each job has to be processed sequentially in three stages. In the first stage, the preprocessing task runs on a master machine; in the second stage, the slave task runs on a dedicated slave machine; and in the last stage, the postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. We use  $a_i$ ,  $b_i$  and  $c_i$  to denote the preprocessing, slave and postprocessing tasks and task times of job  $i$ , respectively. We assume that  $a_i > 0$ ,  $b_i > 0$  and  $c_i > 0$ .

A job may have a release time  $r_i \geq 0$ , i.e.,  $a_i$  cannot start until  $r_i$ . In this paper, we assume every job has release time 0, unless stated otherwise. There are two cases when arbitrary release time is present. The first case deals with offline problems, i.e., the release times and processing times of all jobs are known in advance. The second case deals with online problems, i.e., we have no knowledge of a job  $i$  until it arrives at  $r_i$ , and when it arrives, we know everything about job  $i$ . We use the quadruple  $(r_i, a_i, b_i, c_i)$  to denote job  $i$ . For simplicity, if  $r_i = 0$ , we use the triplet  $(a_i, b_i, c_i)$  to represent job  $i$ .

Each machine is either a master machine or a slave machine. The master machines are used to run preprocessing and/or postprocessing tasks, and the slave machines are used to run slave tasks, one slave machine for each slave task. In a single-master system, there is a single master to execute all preprocessing tasks ( $a$  tasks) and postprocessing tasks ( $c$  tasks). In a multi-master system, there are more than one master, each of which is capable of processing both  $a$  tasks and  $c$  tasks. Finally, in some systems, there are distinct preprocessing masters (preprocessors) and postprocessing masters (postprocessors), which are used to process  $a$  tasks and  $c$  tasks, respectively.

\*This work was partially supported by NSF Grant DMI-0300156.

The master-slave model is closely related to the flow shop model. The system which has a single preprocessing master and a single postprocessing master can be seen as a two-machine flow shop with transfer lags ([17]). When there are more than one preprocessing and postprocessing masters, the model is a two-stage hybrid flow shop with transfer lags. In this sense, we can regard the previous case as a three-stage hybrid flow shop where the first and the last stage has a single machine and the second stage has  $n$  machines. In recent years, hybrid flow shop has received significant attention due to its applications, see [1], [9], [10] and [16].

The master-slave model finds many applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc. For more applications, see [13], [14], [15] and [18].

## 1.1 Preliminaries

Given a set of jobs in the master-slave system and a schedule  $S$  of the jobs, the completion (or finish) time of job  $i$  in  $S$  is the time when the postprocessing task  $c_i$  finishes. We denote the completion time of  $i$  in  $S$  by  $C_i(S)$ . If  $S$  is clear from the context, we use  $C_i$  instead of  $C_i(S)$ . The *makespan* of  $S$  is the earliest time when all the tasks have been completed. We denote the makespan of  $S$  by  $C_{\max}(S)$ , or  $C_{\max}$  if  $S$  is clear from the context. The *total completion time* of  $S$ , denoted by  $C(S)$ , is the sum of the completion times of all  $n$  jobs, i.e.,  $C(S) = \sum_{j=1}^n C_j$ . Makespan and total completion time are two common objectives to minimize. Throughout this paper we use  $C_{\max}^*$  and  $C^*$  to denote the minimum makespan and the minimum total completion time, respectively. Without loss of generality, we always assume that the  $b$  tasks are scheduled immediately as soon as the corresponding  $a$  tasks complete.

An  $\alpha$ -*approximation* algorithm for makespan (or total completion time) is an algorithm that for any set of jobs generates a schedule  $S$  whose makespan (or total completion time) is at most  $\alpha$  times the optimal makespan (or total completion time). It is an  $(\alpha, \beta)$ -*approximation* algorithm if it is an  $\alpha$ -approximation algorithm for makespan and at the same time a  $\beta$ -approximation algorithm for total completion time. For a schedule  $S$ , if  $C_{\max}(S) \leq \alpha C_{\max}^*$  and  $C(S) \leq \beta C^*$ , then we say that  $S$  is an  $(\alpha, \beta)$ -schedule.

The Shortest-Processing-Time (SPT) rule, and the Shortest-Remaining-Processing-Time (SRPT) rule are two well-known algorithms for minimizing total completion time. Suppose each job consists of a single task. If all jobs are available at time 0, then the SPT rule is optimal for total completion time in the single-machine or multi-machine environment. If the release times are arbitrary, then the SRPT rule is optimal for a single machine and it is a 2-approximation (see [12]) in the multi-machine environment.

In this paper, we *adapt* the above two rules for our setting. We apply both rules to generate *preemptive* schedules. A scheduling decision is made when a task completes so that a master machine becomes free, or when a new  $a$  task or  $c$  task becomes available. At any such time instant, the SPT rule schedules, from the set of available tasks (including those that have been preempted but have not yet completed), the one with the smallest processing time. Depending on how we choose from the set of available jobs, we have the  $SPT_a$  rule and the  $SPT_c$  rule. Specifically, in the  $SPT_a$  rule, preemption occurs only among the  $a$  tasks. Similarly we have  $SPT_c$  rule. On the other hand, the SRPT rule schedules, from the set of available tasks, the one with the smallest remaining processing time. We define the  $SRPT_a$  rule and the  $SRPT_c$  rule similarly as above. Both the SPT rule and the SRPT rule may generate schedules with *migration* when there are multiple machines, i.e., after interrupted on one machine, a task is resumed on a different machine.

In this paper, we frequently sort the jobs in certain nondecreasing order of some parameters  $x_i$  of job  $i$ , e.g.  $x_i = c_i$ . For convenience, we say that  $x_i < x_j$  as long as  $x_i$  comes before  $x_j$  in the sorted order, even though it may be the case that  $x_i = x_j$ .

Given a set of jobs in the master-slave system, we may have some constraints or requirements about how these jobs should be scheduled. In the following, we discuss these constraints in detail.

**Canonical schedule versus non-canonical schedule.** A *canonical* schedule on the single master system is one such that all the preprocessing tasks have to finish before any postprocessing tasks can start (Note that our definition of canonical schedule is slightly different from the one given in [13]). In the multi-master system, a canonical schedule is one that is canonical on each master.

Both canonical and non-canonical schedules have some applications. For both makespan and total completion time problems, it has been shown that the problems are NP-hard for both canonical and non-canonical schedules. It is easy to see that for a single master, one can always arrange a schedule to be canonical without increasing the makespan. Thus, for the problem of minimizing the makespan, we only need to focus on canonical schedules. However, for the problem of minimizing total completion time, the ratio of the total completion time of the best canonical schedule versus that of the best non-canonical schedule can be arbitrarily large.

**Preemptive schedule versus non-preemptive schedule.** For both makespan and total completion time problems, it has been shown that the problems are NP-hard even if preemption is allowed. For some instances, preemption does not help to reduce the makespan or total completion time. However, in most cases, preemption can reduce the total completion time and makespan.

## 1.2 Related works

So far the main research efforts to the master-slave model are for minimizing makespan and for special cases of total completion time. The general makespan problem has been shown to be NP-hard by Kern and Nawijn [8]. Leung and Zhao [11] strengthened the result and showed that the problem remains NP-hard in the strong sense, even if all the preprocessing and postprocessing tasks have unit time. In other words, the problem remains NP-hard in the strong sense even if preemption is allowed.

Sahni and Vairaktarakis [14], [18] proposed several heuristics for the makespan problem in the single-master, multi-master and distinct preprocessor and postprocessor systems. The total completion time minimization problem is considered only in [11] by Leung and Zhao. They showed that the problem of finding the optimal canonical or non-canonical schedule is NP-hard even if all preprocessing and postprocessing tasks have unit time. They designed efficient approximation algorithm to minimize the total completion time of canonical schedules on a single-master system when  $a_i = a$  and  $c_i = c$  for all jobs  $i$ .

For the two-stage flow shop with transfer lags model, some research has been done, most of which is about makespan minimization. Dell'Amico [4] proved that the makespan problem is NP-hard, even if preemption is allowed and each stage has only one machine. Later, Yu, Hoogeveen and Lenstra [19] showed that the problem is NP-hard even if all tasks have unit length. This is in contrast to the fact that the problem is solvable in polynomial time when there is no transfer lags. As we mentioned before, this model is the same as the master-slave model when the preprocessing and postprocessing masters are distinct. Thus, the heuristics given in [18] for the master-slave model also work here. Little is known about the total completion time minimization problem.

## 1.3 New results

In this paper, our main objective is to minimize total completion time under various scenarios. This includes single-master systems, multi-master systems, and distinct preprocessing and postprocessing master systems. Since the problem is strongly NP-hard, we can only hope to have approximation algorithms. For each type of system, we try to approximate the best preemptive



Table 1. New results for single-master system.

|              | preemptive                          | non-preemptive                  |
|--------------|-------------------------------------|---------------------------------|
| $r_i = 0$    | $(\frac{3}{2}, 2)$ , canonical      | $(\frac{5}{2}, \frac{2e}{e-1})$ |
| $r_i \geq 0$ | $(2, 2)$ , online and non-canonical | $(3, 4)$                        |

Table 2. New results for multi-master system, all schedules are non-canonical.

|              | preemptive                           | non-preemptive |
|--------------|--------------------------------------|----------------|
| $r_i = 0$    | $(3, 2)$ , no migration              | $(5, 4)$       |
| $r_i \geq 0$ | $(3, 2)$ , offline with no migration | $(5, 4)$       |
| $r_i \geq 0$ | $(3, 2)$ , online with migration     | -              |

Table 3. New results for distinct preprocessor and postprocessor,  $m_1 = m_2 = 1$ .

|              | preemptive         | non-preemptive            |
|--------------|--------------------|---------------------------|
| $r_i = 0$    | $(3, 2)$           | $(4, \frac{2e}{e-1})$     |
| $r_i = 0$    | $(3, 2)$ , offline | $(4, 4 + \frac{2e}{e-1})$ |
| $r_i \geq 0$ | $(3, 2)$ , online  | $(4, 4 + \frac{2e}{e-1})$ |

and non-preemptive schedules. We consider both the case when all jobs have the same release time and the case when the jobs have different release times.

Preemptive relaxation is an important technique for getting constant-factor approximations for total completion time of non-preemptive schedules; see [12], [2], and [6]. The advantage of preemptive relaxation is that usually there are very efficient algorithms to generate optimal or near optimal schedules. In most cases, these algorithms (both preemptive and non-preemptive) can be implemented to run in an online fashion, see [12].

In this paper, we use preemptive relaxation. We first develop efficient algorithms to generate preemptive offline or online schedules with good approximation. We show that these schedules have small makespan as well. Then, by applying the ideas in [12] and [3] to our models, we convert these preemptive schedules into non-preemptive schedules with certain degradation in the quality of approximation. Our results are summarized in Tables 1, 2, 3 and 4. In these tables,  $e$  denotes the base of natural logarithm.

The organization of this paper is as follows. In Sections 2, 3 and 4, we consider preemptive schedules. Section 2 is devoted to the single-master case, Section 3 is devoted to the multi-master case, and Section 4 is devoted to the case of distinct preprocessor and postprocessor. In Section 5, we consider the conversions from preemptive schedules into non-preemptive schedules. Finally, we draw some conclusions in the last section.

Table 4. New results for distinct preprocessor and postprocessor,  $m_1 \geq 1$  and  $m_2 \geq 1$ .

|              | preemptive                        | non-preemptive      |
|--------------|-----------------------------------|---------------------|
| $r_i = 0$    | $(4, 2)$ , with migration         | $(4, \frac{14}{3})$ |
| $r_i \geq 0$ | $(4, 3)$ , offline with migration | $(5, 13)$           |
| $r_i \geq 0$ | $(4, 3)$ , online with migration  | $(5, 13)$           |

## 2. Single-master systems

In this section we assume that there is a single master. We first study canonical schedules, and then non-canonical schedules. Finally, we study the case when jobs have different release times.

### 2.1 Canonical preemptive schedules

We begin with two lower bounds of canonical schedules. Suppose we are given  $n$  jobs  $1, 2, \dots, n$ . Throughout this paper, we let  $A = \sum_{j=1}^n a_j$ ,  $B = \sum_{j=1}^n b_j$  and  $C = \sum_{j=1}^n c_j$ . For canonical schedules, preemption allowed or not, we have the following lower bound

$$C^* \geq nA + \sum_{j=1}^n \sum_{c_i \leq c_j} c_i, \quad (1)$$

which assumes that there is no idle time in the schedule and that the  $c$  tasks are scheduled in ascending order of their lengths. Another lower bound is

$$C^* \geq \sum_{j=1}^n \sum_{a_i \leq a_j} a_i + B + C, \quad (2)$$

which assumes that jobs are scheduled in increasing order of the  $a_i$ , and that the  $b$  tasks and the  $c$  tasks are scheduled immediately after they are available. Finally, we have the following trivial lower bound for any schedules which is simply the summation of all the processing times.

$$C^* \geq A + B + C. \quad (3)$$

In canonical schedules, all the  $a$  tasks are scheduled first. Since all the  $a$  tasks are available at time 0 and the  $c$  tasks cannot start until all the  $a$  tasks finish, there is no need to preempt the  $a$  tasks. Hence, only a  $c$  task can be preempted by another  $c$  task in this case.

**Heuristic 1.:** Schedule the  $a$  tasks in an arbitrary order. After all the  $a$  tasks finish, schedule the available  $c$  tasks by the  $SPT_c$  rule.

**THEOREM 1** *Heuristic 1 generates a (2, 2) canonical preemptive schedule in  $O(n \log n)$  time.*

**Proof :** Let  $C_{a_j}$  denote the time  $a_j$  completes. Then at time  $t_j = \max(A, (C_{a_j} + b_j))$ , all the  $a$  tasks finish and the  $c_j$  is available to be scheduled. Since  $C_{a_j} \leq A$ , we have  $t_j \leq A + b_j$ . According to the heuristic, if there is another available task  $c_i$  that hasn't finished at time  $t_j$  and  $c_i < c_j$ , then  $c_j$  has to wait until  $c_i$  finishes. Also, during the execution of  $c_j$ , if there is another task  $c_i < c_j$  that becomes available, then  $c_i$  preempts  $c_j$ . In both cases, we say that  $c_j$  is *delayed* by  $c_i$ . Let  $C_j$  be the completion time of  $c_j$  in the schedule generated by Heuristic 1. Then, we have

$$C_j = t_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \leq (A + b_j + c_j) + \sum_{c_i < c_j} c_i.$$

The total completion time is

$$\sum_{j=1}^n C_j \leq \sum_{j=1}^n \left( A + b_j + c_j + \sum_{c_i < c_j} c_i \right) = \left( nA + \sum_{j=1}^n \sum_{c_i < c_j} c_i \right) + (B + C) < 2C^*,$$

where the last inequality comes from the lower bounds (1) and (3).

It has been shown in [14] that any canonical schedule without preemption is a 2-approximation for makespan. Since preemption among the  $c$  tasks can not increase the makespan, the schedule generated by Heuristic 1 has makespan at most two times the optimal. Together with the above bound of total completion time, this concludes the theorem.  $\square$

In [14], an algorithm is developed which generates schedule with makespan at most  $3/2$  times the optimal schedule. Thus in Heuristic 1, if we schedule the  $a$  tasks in the same order as the algorithm in [14], we will get a  $3/2$ -approximation for makespan, since preemption on the available  $c$  tasks will not increase the makespan.

**COROLLARY 2** *There is an  $O(n \log n)$  time algorithm that generates a  $(3/2, 2)$  canonical preemptive schedule.*

Note that when preemption occurs we use the SPT rule instead of the SRPT rule. This is for the purpose of analysis only. In practice, one can use the SRPT rule to get a better approximation for total completion time.

## 2.2 Non-canonical preemptive schedules

In this model, the  $a$  tasks and the  $c$  tasks can be scheduled alternatively. A lower bound on the  $C^*$  can be obtained by assuming all the  $b$  tasks have length 0:

$$C^* \geq \sum_{j=1}^n \sum_{a_i + c_i \leq a_j + c_j} (a_i + c_i) \quad (4)$$

**Heuristic 2:.** For any two jobs, if  $(a_j + c_j) < (a_i + c_i)$ , then we say that both  $a_j$  and  $c_j$  have higher priority than  $a_i$  and  $c_i$ . At any time, if the master processor is free for assignment, assign the available task with the highest priority. If a new task becomes available and has higher priority than the currently running task, the new task preempts the currently running task.

**THEOREM 3** *Heuristic 2 generates a  $(2, 2)$  preemptive schedule for a single-master system when all jobs have the same release times.*

**Proof :** Due to space limit, we sketch the proof. Note that an  $a$  task or  $c$  task can be delayed only by those tasks with higher priority. Using the lower bounds (4) and (3), one can show that the total completion time is at most two times the optimal. For the makespan, pick the last job  $j$  such that  $c_j$  is scheduled immediately when it is ready. Such a job always exists since the job  $i$  with the highest priority must be such a job. Then, the interval  $I_1 = [0, C_{a_j})$  and the interval after  $c_j$  starts till the end, i.e.,  $I_2 = [(C_{a_j} + b_j), C_{\max})$ , contain no idle time. Denote their lengths by  $|I_1|$  and  $|I_2|$ , respectively. Then  $|I_1| + |I_2| \leq C_{\max}^*$ . Thus, the makespan is

$$C_{\max} = |I_1| + b_j + |I_2| \leq C_{\max}^* + b_j < 2C_{\max}^*$$

$\square$

## 2.3 Arbitrary release times

When arbitrary release times are present, it is not meaningful to have canonical schedules any more. Thus, we consider only non-canonical schedules. Apparently, the lower bound (4) still holds for this case. Let  $R = \sum_{i=1}^n r_i$ . Then, a trivial lower bound for the minimum total completion time of any schedule is

$$C^* \geq \sum_{i=1}^n (r_i + a_i + b_i + c_i) = R + A + B + C \quad (5)$$

Observe that Heuristic 2 makes no assumptions about the release time of a job. So, we can still apply Heuristic 2 when jobs have arbitrary release times. Unlike the case when all jobs have the same release times, in this case, a higher priority  $a$  task may also delay a lower priority  $a$  task or  $c$  task. Using similar argument, one can prove the following theorem.

**THEOREM 4** *Heuristic 2 generates a  $(2, 2)$  preemptive schedule online for a single-master system even when the jobs have arbitrary release times.*

### 3. Multi-master systems

In this section, we assume that there are  $m \geq 2$  masters, each of which is capable of processing both the  $a$  tasks and the  $c$  tasks.

#### 3.1 Non-canonical preemptive schedules

Let us assume that the jobs are indexed in *nondecreasing* order of  $a_j + c_j$ . That is,  $a_j + c_j \leq a_{j+1} + c_{j+1}$  for  $1 \leq j \leq n - 1$ .

**Heuristic 3.:** (1) Without loss of generality, we may assume that  $n$  is a multiple of  $m$ . Otherwise, we add dummy jobs with  $a_i = b_i = c_i = 0$ . (2) We assign the jobs to the machines such that jobs  $1, 2, \dots, m$  go to machines  $1, 2, \dots, m$ , respectively; jobs  $m + 1, m + 2, \dots, 2m$  go to machines  $m, m - 1, \dots, 1$ , respectively; jobs  $2m + 1, 2m + 2, \dots, 3m$  go to machines  $1, 2, \dots, m$ , respectively; and so on until all jobs are assigned. (3) Apply Heuristic 2 to each master machine to schedule the jobs assigned to it.

**THEOREM 5** *Heuristic 3 generates a  $(3, 2)$  preemptive schedule for multi-master systems without migration when all jobs have release time 0.*

**Proof :** Without loss of generality, we may assume that  $n = mk$  for some integer  $k$ . For convenience, we reindex the jobs assigned to each machine  $p$  in the form of  $(p, q)$  such that  $a_{(p,q)} + c_{(p,q)} \leq a_{(p,q+1)} + c_{(p,q+1)}$ . A lower bound of the total completion time comes from the fact that Heuristic 3 is optimal if  $b_{(p,q)} = 0$  for every job  $(p, q)$ .

$$C^* \geq \sum_{p=1}^m \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \quad (6)$$

Fix a machine  $p$ . Let  $C_{(p,q)}$  denote the completion time of job  $(p, q)$  and  $B_p = \sum_{q=1}^k b_{(p,q)}$ . Following the analysis of Heuristic 2 in Section 1.2.2, we have

$$\sum_{q=1}^k C_{(p,q)} \leq \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B_p .$$

Thus, the total completion time is

$$\begin{aligned} \sum_{p=1}^m \sum_{q=1}^k C_{(p,q)} &\leq \sum_{p=1}^m \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) + B_p \right) \\ &\leq \sum_{p=1}^m \left( \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B \quad \text{by (6) and (3)} \\ &< 2C^* . \end{aligned}$$

Now we analyze the makespan of the schedule. Let  $\Delta = A + C$ . Then,

$$C_{\max}^* \geq \frac{\sum_{j=1}^n (a_j + c_j)}{m} = \frac{\Delta}{m} .$$

Let  $\Delta_p = \sum_{j \text{ scheduled on } p} (a_j + c_j)$ . By the pigeon hole principle,  $\min_{1 \leq p \leq m} \Delta_p \leq \Delta/m < C_{\max}^*$ . For any two machines  $p$  and  $q$ , by the way we assign jobs to the machines

$$\Delta_p - \Delta_q \leq \max_{1 \leq k \leq n} (a_k + c_k) - \min_{1 \leq k \leq n} (a_k + c_k) \leq \max_{1 \leq k \leq n} (a_k + c_k) < C_{\max}^* .$$

This means that for any machine  $p$ ,  $\Delta_p \leq \min_{1 \leq q \leq m} \Delta_q + C_{\max}^* \leq 2 C_{\max}^*$ . Suppose the job with the maximum completion time among all jobs is assigned to machine  $p$ . Let  $l$  be the last job on machine  $p$  so that  $c_l$  is scheduled immediately after it is ready. Define  $C_{a_l}$  as before. Then machine  $p$  is busy during the intervals  $I_1 = [0, C_{a_l}]$  and  $I_2 = [(C_{a_l} + b_l), C_{\max}]$ . The total length of the two intervals is  $|I_1| + |I_2| \leq \Delta_p < 2 C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = |I_1| + b_l + |I_2| < 3 C_{\max}^* .$$

□

### 3.2 Arbitrary release times

**Offline schedule without migration.** In this case, we can still apply Heuristic 3. However, note that to assign jobs to machines, Heuristic 3 requires that we have full knowledge of all the jobs at the beginning. Thus, Heuristic 3 is an offline algorithm. Combining the arguments in Sections 2.3 and 3.1, we have the following theorem.

**THEOREM 6** *Heuristic 3 generates a (3, 2) offline preemptive schedule without migration for multi-master systems when jobs have arbitrary release times.*

**Online schedule with migration.** In this case, we apply Heuristic 2 to multi-master systems. Note that if a new task becomes available and one or more currently running tasks have lower priority than the new task, then the task with the lowest priority will be preempted.

**THEOREM 7** *Heuristic 2 generates a (3, 2) online preemptive schedule with migration on multi-master systems when jobs have arbitrary release times.*

**Proof :** (sketch). First one can easily show the following bound:

$$C^* \geq \sum_{j=1}^n \sum_{a_i + c_i \leq a_j + c_j} (a_i + c_i) / m . \quad (7)$$

Let  $C_{a_j}$  denote the completion time of  $a_j$  in the schedule generated by Heuristic 2. Let  $I_1 = [r_j, (C_{a_j} - a_j))$  and  $I_2 = [C_{a_j} + b_j, (C_j - c_j))$ . Then one can show there is no idle time during these two intervals. Furthermore, only tasks with higher priority than  $j$  can be scheduled during this intervals. Let  $|I_1|$  and  $|I_2|$  denote the lengths of  $I_1$  and  $I_2$ , respectively. Then, we must have  $|I_1| + |I_2| \leq \sum_{a_i + c_i < a_j + c_j} (a_i + c_i) / m$ . Thus, we can bound the completion time  $C_j$  as follows:

$$\begin{aligned} C_j &\leq r_j + |I_1| + a_j + b_j + |I_2| + c_j \\ &\leq \left( \sum_{a_i + c_i < a_j + c_j} (a_i + c_i) / m \right) + (r_j + a_j + b_j + c_j) . \end{aligned}$$

Summing up all  $C_j$  together and using the bounds (7) and (5), we can show that  $\sum_{j=1}^n C_j < 2 C^*$ . For the makespan of the schedule, let  $k$  be the job with the maximum completion time among all jobs. Let  $l$  be the last job among all jobs so that  $c_l$  is scheduled immediately after it is ready. Define  $C_{a_l}$  as before. Then, all machines must be busy during the intervals  $I_1 = [r_l, C_{a_l} - a_l)$  and  $I_2 = [(C_{a_l} + b_l), C_{\max} - c_k)$ . The total length of the two intervals is  $|I_1| + |I_2| \leq C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + a_l + b_l + |I_2| + c_k = (r_l + a_l + b_l) + (|I_1| + |I_2|) + c_k < 3 C_{\max}^* .$$

□

#### 4. Distinct preprocessing and postprocessing masters

In this section, we study the model with distinct preprocessing masters and postprocessing masters. Different from the previous cases, here an  $a$  task can only be preempted by another  $a$  task and a  $c$  task can only be preempted by another  $c$  task. In all cases, we apply the following heuristic to obtain a preemptive schedule.

**Heuristic 4:** Schedule the available  $a$  tasks using the  $SRPT_a$  rule on the preprocessors; Schedule the available  $c$  tasks using the  $SPT_c$  rule on the postprocessors.

Let  $m_1$  and  $m_2$  denote the numbers of preprocessing masters and postprocessing masters, respectively. We first study the simple case  $m_1 = m_2 = 1$ .

**THEOREM 8** *Heuristic 4 generates a (3, 2) online preemptive schedule when  $m_1 = m_2 = 1$  and jobs have the same or arbitrary release times.*

**Proof :** We first consider the total completion time. Let  $C_{a_j}$  be the time  $a_j$  finishes in the schedule obtained by Heuristic 4. The key observation is that our schedule has the minimum  $\sum_{j=1}^n C_{a_j}$  among all possible schedules. Let  $C_{a_j}^*$  be the time  $a_j$  finishes in an optimal schedule. Thus, we have

$$C^* \geq \sum_{j=1}^n (C_{a_j}^* + b_j + c_j) = \left( \sum_{j=1}^n C_{a_j}^* \right) + B + C > \left( \sum_{j=1}^n C_{a_j} \right) + B + C = \sum_{j=1}^n (C_{a_j} + b_j + c_j) .$$

Notice  $c_j$  can only be delayed by smaller  $c$  tasks. Thus, the total completion time of the schedule obtained by Heuristic 4 is at most

$$\sum_{j=1}^n \left( C_{a_j} + b_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \right) \leq \sum_{j=1}^n (C_{a_j} + b_j + c_j) + \sum_{j=1}^n \sum_{c_i < c_j} c_i \leq 2 C^* .$$

For the makespan, consider the last job  $l$  such that  $c_l$  runs immediately when it is available at time  $C_{a_l} + b_l$ . There is no idle time in the interval  $I_1 = [r_l, C_{a_l})$  and the interval  $I_2 = [(C_{a_l} + b_l), C_{\max})$ . The length of each interval is at most  $C_{\max}^*$ . Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + b_l + |I_2| \leq 3 C_{\max}^* .$$

□

**THEOREM 9** *When  $m_1 > 1$  and  $m_2 > 1$ , Heuristic 4 generates a (4, 2) preemptive schedule with migration if the jobs have the same release times and a (4, 3) preemptive schedule with migration if the jobs have arbitrary release times.*

**Proof :** Using similar argument as in the previous theorem, and the fact that  $SRPT_a$  rule minimize the total completion time of  $C_{a_j}$  when all jobs have the same release time and that it is a 2-approximation when the jobs have arbitrary release times in the multi-machine environment. □

## 5. Converting preemptive schedules into non-preemptive schedules

As we mentioned before, we obtain non-preemptive schedules by converting from preemptive schedules. Our approach is based on the technique that was introduced by Phillips et. al in [12], and improved by Chekuri et. al in [3]. The model studied in [12] consists of one or more identical machines and  $n$  simple jobs. Let  $S$  be a preemptive schedule. To obtain a non-preemptive schedule  $S'$ , they form a list of jobs in increasing order of their completion times in  $S$  and then list schedule the jobs in this list one by one, respecting their release times. They showed that if  $S$  is a  $\beta$ -approximation for total completion time, then  $S'$  is a  $2\beta$ -approximation for total completion time in the single-machine environment, and a  $3\beta$ -approximation for total completion time in the multi-machine environment. In addition, this conversion also yields an online non-preemptive algorithm if the preemptive schedule can be generated online.

Later, Chekuri et. al [3] improved the above results in the single machine case. The difference between the two approaches in [3] and [12] is how to obtain the list of jobs. Chekuri et. al designed a deterministic  $O(n^2)$  time *offline* algorithm such that the schedule obtained has total completion time at most  $\frac{e}{e-1}$  times that of the preemptive schedule, where  $e$  is the base of natural log. As well, they gave a randomized *online* algorithm with *expected* performance  $\frac{e}{e-1}$ . In the multi-machine case, Chakrabarti et al. [2] showed that the convert procedure given in [12] has a bound of  $7/3$ , instead of 3 times that of  $S$ . In both cases one can show that the makespan of the  $S'$  is not much larger than that of  $S$ .

In the following sections, we will describe how to convert preemptive schedules generated by Heuristics 1-4 to non-preemptive schedules in the master-slave model. The difficulty of our conversion is that we need to respect not only the release time of  $a_i$ ,  $1 \leq i \leq n$ , but also respect the constraint that the interval between the finish time of  $a_i$  and the start time of  $c_i$  has length at least  $b_i$ . By our convention, we use  $S$  to denote a  $(\alpha, \beta)$  preemptive schedule,  $C_{a_j}$  to denote the completion time of  $a_j$  in  $S$ ,  $C_j$  to denote the completion time of  $j$  in  $S$ , and  $C_{\max}$  to be the makespan of  $S$ . Our goal is to convert  $S$  into a non-preemptive schedule  $S'$ . We define  $C'_{a_j}$ ,  $C'_j$ ,  $C'_{\max}$  similarly for the non-preemptive schedule  $S'$ .

### 5.1 Single master and multi-master systems

For a canonical schedule  $S$  on a single machine, all jobs have the same release time and preemption occurs only among the  $c$  tasks. We can think of the  $c$  tasks as having release time  $\max(A, (C_{a_j} + b_j))$  with processing time  $c_j$ . Thus, we can do the conversion for the  $c$  tasks using the approach of [3], respecting the release time  $\max(A, (C_{a_j} + b_j))$ . The obtained schedule is a  $(1 + \alpha, \frac{e}{e-1}\beta)$ -schedule.

If we take  $S$  to be the  $(\frac{3}{2}, 2)$  canonical schedule in Corollary 2 in Theorem 3 and apply the above conversions, we have the following theorem.

**THEOREM 10** *In  $O(n^2)$  time, one can obtain a  $(\frac{5}{2}, \frac{2e}{e-1})$  non-preemptive canonical schedule when there is a single master and all jobs have the same release time.*

To get a non-canonical schedule without preemption, we let  $S$  to be the preemptive  $(2, 2)$  non-canonical schedule in Theorem 4. Then we obtain a  $\lambda$ -schedule  $S'$ ,  $\lambda = 1$ . Using similar argument as [12], one can show that  $S'$  is a  $(3, 4)$ -schedule. Furthermore, we can generate  $S'$  in an online fashion. Thus we have the following theorem.

**THEOREM 11** *In  $O(n \log n)$  time, one can obtain a  $(3, 4)$  online non-preemptive schedule when there is a single master and  $r_j \geq 0$  for all job  $j$ .*

The following theorem shows how to get offline non-preemptive schedules for multi-master systems.

**THEOREM 12** *When there are multi-masters, one can obtain a  $(5, 4)$  non-preemptive offline schedule.*

**Proof :** Let  $S$  be the (3,2)-schedule generated by Heuristic 3. Then  $S$  has no migration. We can obtain the non-preemptive schedule  $S'$  by converting the schedule on each machine separately in the same way as described in the proof of Theorem 11. Thus, the total completion time of  $S'$  is at most two times that of  $S$ . Since  $S$  is a 2-approximation for total completion time,  $S'$  is a 4-approximation for total completion time. For the makespan,  $C'_j \leq C_j + \max_{i=1}^m \Delta_i$  where  $\Delta_i$  is the total length of the  $a$  tasks and the  $c$  tasks assigned on machine  $i$ . We have shown in Section 1.3 that  $\Delta_i \leq 2C_{\max}^*$ . Thus,  $C'_j \leq C_j + \max_{i=1}^m \Delta_i \leq C_j + 2C_{\max}^*$ . Since  $S$  is 3-approximation for makespan,  $S'$  is a 5-approximation for the makespan. This concludes the proof.  $\square$

## 5.2 Distinct preprocessors and postprocessors

**THEOREM 13** *In  $O(n^2)$  time, one can obtain a  $(4, \frac{2e}{e-1})$  non-preemptive schedule when  $r_j = 0$  for all  $j$ , and  $m_1 = m_2 = 1$ .*

**Proof :** Since all jobs have the same release time, we can assume there is no preemption on the single preprocessor. So we simply do conversion on the single postprocessor using the approach given in [3]. Let  $S$  be the (3,2)-schedule generated by Heuristic 4. Then  $S'$  is a  $(4, \frac{2e}{e-1})$  non-preemptive schedule.  $\square$

When the release times are arbitrary, we need to do the conversion carefully so as to make sure that the difference between the finish time of  $a_j$  and the start time of  $c_j$  is at least  $b_j$ .

**THEOREM 14** *When  $m_1 = m_2 = 1$ , and  $r_j \geq 0$ , one can obtain a  $(4, 4 + \frac{2e}{e-1})$  non-preemptive offline schedule or an online non-preemptive schedule with expected performance  $(4, 4 + \frac{2e}{e-1})$ .*

**Proof :** We first remove the preemptions among the  $a$  tasks to get a  $\lambda$ -schedule of  $a$  tasks, respecting the release times of the  $a$  tasks. Let  $C'_{a_j}$  be the new completion time of  $a_j$ . Then, we must have  $C'_{a_j} \leq \frac{e}{e-1} C_{a_j}$ , by [3]. Next, we remove the preemptions among the  $c$  tasks to get a  $\lambda$ -schedule of the  $c$  tasks, where  $\lambda = 1$ , and make sure the interval between  $C'_{a_j}$  and the start time of  $c_j$  is at least  $b_j$  for each job  $j$ .

Let  $C_j$  be the completion time in the preemptive schedule  $S$ . Suppose the jobs are indexed such that  $C_i < C_{i+1}$ . Then,  $C_j \geq \max_{i=1}^j r_i$ ,  $C_j \geq \max_{i=1}^j b_i$ ,  $C_j > \sum_{i=1}^j a_i$  and  $C_j > \sum_{i=1}^j c_i$ .

Let  $C'_j$  be the new completion time of job  $j$  in  $S'$ . Then we have

$$C'_j \leq C'_{a_j} + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^j c_i \leq (2 + \frac{e}{e-1}) C_j .$$

For the makespan, we have

$$C'_j \leq \max_{1 \leq i \leq j} r_i + \sum_{C_{a_i} \leq C_{a_j}} a_i + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^j c_i \leq 4 C_{\max}^* .$$

Let  $S$  be the (3,2)-schedule generated by Heuristic 4. Then the theorem follows.  $\square$

**THEOREM 15** *In  $O(n \log n)$  time, one can obtain a  $(4, 14/3)$  non-preemptive schedule when  $r_j = 0$  for all  $j$ ,  $m_1 \geq 1$  and  $m_2 \geq 1$ .*

**Proof :** Since all jobs have the same release time, we can assume there is no preemption on the preprocessors. So we do conversion on the postprocessors using the approach given in [2]. Let  $S$



be the  $(4, 2)$ -schedule generated by Heuristic 4. Then the total completion time of  $S'$  is at most  $\frac{7}{3}$  times that of  $S$ , i.e.,  $S'$  is a  $14/3$ -approximation for total completion time.

We now consider the makespan. Suppose the jobs are indexed such that  $C_i \leq C_{i+1}$  in  $S$ . By the heuristic, for any  $1 \leq i \leq n$ ,  $C_{a_i} + b_i \leq \sum_{k \neq i} a_k/m_1 + (a_i + b_i) \leq 2 C_{\max}^*$  and  $\sum_{i=1}^j c_i/m_2 \leq C_{\max}^*$ . In  $S'$ , the latest time the postprocessors become busy is  $\max_{i \leq j} (C_{a_i} + b_i)$ . Therefore, we have  $C'_j \leq \max_{i \leq j} (C_{a_i} + b_i) + \sum_{i=1}^{j-1} c_i/m_2 + c_j \leq 4C_{\max}^*$   $\square$

**THEOREM 16** *In  $O(n \log n)$  time, one can obtain a  $(5, 13)$  non-preemptive online schedule when  $m_1 \geq 1$  and  $m_2 \geq 1$ .*

**Proof :** We first remove preemptions among the  $a$  tasks to get a  $\lambda$ -schedule of the  $a$  tasks, respecting the release times of the  $a$  tasks. Let  $C'_{a_j}$  be the new completion time of  $a_j$ . Then, we must have  $C'_{a_j} \leq (\frac{7}{3}) C_{a_j}$ , by the result of [2]. Next, we remove preemptions among the  $c$  tasks to get a  $\lambda$ -schedule of the  $c$  tasks, where  $\lambda = 1$ , and make sure that the interval between  $C'_{a_j}$  and the start time of  $c_j$  is at least  $b_j$  for each job  $j$ .

Let  $C_j$  be the completion time in the preemptive schedule. Suppose the jobs are indexed such that  $C_i < C_{i+1}$  in  $S$ . Then,  $C_j \geq \max_{1 \leq i \leq j} (C_{a_i} + b_i + c_i)$  and  $C_j > \sum_{i=1}^j c_i/m_2$ . Let  $C'_j$  be the new completion time of job  $j$ . Then we have

$$C'_j \leq \max_{1 \leq i \leq j} (C'_{a_i} + b_i) + \sum_{1 \leq i < j} c_i/m_2 + c_j \leq \max_{1 \leq i \leq j} (\frac{7}{3} C_{a_i} + b_i) + \sum_{1 \leq i < j} c_i/m_2 + c_j \leq \frac{13}{3} C_j .$$

For the makespan, we have  $C_{\max}^* \geq A/m_1$ ,  $C_{\max}^* \geq C/m_2$  and  $C_{\max}^* \geq a_j + b_j + c_j$ . Therefore, we have

$$\begin{aligned} C'_j &\leq \max_{1 \leq k \leq n} r_k + \left( \sum_{C_{a_k} < C_{a_j}} a_k/m_1 \right) + a_j + \max_{C_i < C_j} b_j + \left( \sum_{1 \leq i < j} c_i/m_2 \right) + c_j \\ &\leq 5 C_{\max}^* \end{aligned}$$

Thus, if  $S$  is a  $(4, 3)$ -schedule generated by Heuristic 4, then  $S'$  is a  $(5, 13)$ -schedule.  $\square$

## 6. Conclusion

In this paper we have considered the problem of minimizing total completion time in the master-slave model in various settings. We designed efficient algorithms to generate preemptive schedules with good performance ratio. In most cases, these schedules have small makespan as well. Then we convert the preemptive schedules into non-preemptive schedules using the techniques developed in [12], [2] and [3].

Several problems remain open. We are not able to find good canonical schedules in multi-master systems. The difficulty is in obtaining a good lower bound of the optimal schedule in this case. In some cases, the non-preemptive schedules obtained by the conversion procedure give us a rather large performance ratio. It is worthwhile to consider using different techniques such as linear programming relaxation. Most of the performance bounds are quite loose. For future research, it is worthwhile to tighten the performance bounds.

## References

- [1] R.E. Buten and V.Y. Shen. A scheduling model for computer systems with two classes of processors. Sagamore Computer Conference on Parallel Processing, pp. 130-138, 1973.

- [2] S. Chakrabarti, C. Phillips, A. Schulz, D.B. Shmoys, C. Stein and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proc. 23rd ICALP*, pp. 646-657, 1996
- [3] C. Chekuri, R. Motwani, B. Natarajan and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1), pp. 146-166, 2001.
- [4] M. Dell'Amico. Shop problems with two machine and time lags. *Operations Research*, 44(5), pp. 777-787, 1996.
- [5] J. Du and J.Y-T. Leung. Minimizing mean flow time in two-machine open shops and flow shops. *Journal of Algorithms*, 14:24-44, 1993.
- [6] M.X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proc. 8th SODA*, pp. 591-598, 1997.
- [7] J.N.D. Gupta. Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 38, pp. 359-364, 1988.
- [8] W. Kern and W. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. University of Twente, 1993.
- [9] M.A. Langston. Interstage transportation planning in the deterministic flow-shop environment. *Operations Research*, 35(4), pp. 556-564, 1987.
- [10] C.-Y. Lee and G.L. Vairaktarakis. Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16, pp. 149-158, 1994.
- [11] J.Y-T. Leung and H. Zhao. Minimizing mean flowtime on master-slave machines. In *Proc. 2004 PDPTA*, Vol. II, pp. 939-945, 2004.
- [12] C. Phillips, C. Stein and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199-223, 1998.
- [13] S. Sahni. Scheduling master-slave multiprocessor systems. *IEEE Trans. on Computers*, 45(10), pp. 1195-1199, 1996.
- [14] S. Sahni and G. Vairaktarakis. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9, pp. 357-377, 1996.
- [15] S. Sahni and G. Vairaktarakis. The master-slave scheduling model. In J. Y-T. Leung (Ed): *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL, 2004.
- [16] C. Sriskandarajah and S.P. Sethi. Scheduling algorithms for flexible flowshops: worst and average case performance. *European Journal of Operational Research*, 43, pp. 143-160, 1989.
- [17] W. Szwarc. Flow-shop problems with time lags. *Management Science*, 29, pp. 447-491, 1983.
- [18] G. Vairaktarakis. Analysis of algorithms for master-slave system. *IIE Transactions*, 29(11), pp. 939-949, 1997.
- [19] W. Yu, H. Hoogeveen and J.K. Lenstra. Minimizing makespan in a two-machine flowshop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5), 333 - 348, 2004.

# **Search Algorithms**

Abstracts

## **Skills-Based Agent Scheduling at Call Centers**

**Turgut Aykin and John S. McKenna**

**ac2 Solutions**

**1 Bethany Rd., Suite: 52**

**Hazlet, NJ 07730**

**[taykin@ac2solutions.com](mailto:taykin@ac2solutions.com)**

**[johnsmckenna@ac2solutions.com](mailto:johnsmckenna@ac2solutions.com)**

This paper provides a review of the approaches available for the agent scheduling problem at a call center. Complexities found in scheduling agents at large call centers are discussed together with the implications of skills-based routing on this problem. Computational results from several large call center environments will be reported.

# Timetable Synchronization for Mass Transit

JANNY M.Y. LEUNG and RACHEL C.W. WONG

*Systems Engineering and Engineering Management Department,  
The Chinese University of Hong Kong  
Shatin, New Territories Hong Kong.*

## Abstract:

Nothing to do but to look for the next transfer train is the passenger's plight when taking public transit in many places. To be able to design timetables with good co-ordination between train-lines so that passengers could enjoy "immediate" transfer is a service goal of the Mass Transit Railway Corporation (MTRC), which runs six railway lines with 13 interchange stations in Hong Kong. Whilst important, this problem has not received widespread research attention and there are few papers studying the problem of minimizing the transfer waiting time in a transit network, mostly by heuristic methods.

In this paper, we propose a mixed integer programming (MIP) optimization model for this timetable synchronization problem. The objective is to minimize the sum of all waiting times of all passengers at interchange stations in a railway system. By adjusting the trains' run-times and station dwell-times during their trips, and their dispatch times, turnaround times and headways at the terminals, we can construct high-quality timetables that optimize the objective of minimizing passenger waiting times. A novelty in our formulation is the use of binary variables to determine the relative sequencing of trains on different lines with passenger transfers, which enables the correct representation of the waiting times for transfers to the "next available" train at interchange stations. Numerical results will be reported, which indicate that our approach improves the synchronization of the current schedule significantly.

With trains departing every few minutes from each terminal, there are a large number of trips to consider, and hence the MIP formulation for the timetable synchronization contains thousands of binary variables and tens of thousands of continuous variables and constraints. We also investigated an optimization-based

heuristic for this problem, where we first solve the LP-relaxation of the MIP formulation. Based on the fractional value of the integer variables, we heuristically “predict” their values, which determine the relative sequencing of the trains on different lines. Once we fix the values of some of the integer variables, we solve the resulting linear programming (relaxation) formulation, which is much smaller than the original MIP, to get the arrival times, dwell times, dispatch times, headways, etc. for all trains in the entire schedule in a very short time. By heuristically searching for the subsets of integer variables to fix, we can get good-quality solutions within a reasonably short time.

We focus our timetable synchronization study on railways only because the run times and dwell times are easier to control. Disturbances (e.g. congestion, traffic accident) in railroad are less than for road-based transport. Synchronized timetables are more easily implemented for railways, and thus resulting benefits are significant. That said, the formulation proposed in this paper is generally applicable to other (possibly multi-modal) transport systems as well. Also in our model, we not only adjust run times and dispatch times of trains but also dwell times, turnaround times and headway of trains, which are not studied in other papers.

In our preliminary study, we consider the train schedule in the MTR system in Hong Kong for both rush-hour and non-rush-hour periods. Using our model formulation, we constructed a schedule that reduce the waiting time for each transferring passenger by over 65% on average, compared to the current schedule. These solutions, whilst not guaranteed optimal, are obtained within a few hours of computation time. These are encouraging results. We also explore the trade-offs among different operational parameters and flexibility and their impact on overall passenger waiting-times.

Mass transit railways are important and growing transportation systems, especially for large metropolitan cities. By 2018, Hong Kong will have 12 railway lines with 26 interchange stations. High quality timetables can conserve time for passengers, generate goodwill, and have other advantages (e.g., reducing the air-conditioning load at stations). There is a need to develop systems to synchronize the schedule of trains.

# Algorithms for online buffer management

Co-authors: Nikhil Bansal, Lisa K Fleischer, Tracy Kimbrel, Baruch Schieber, and Maxim Sviridenko, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.

Mohammad Mahdian, Microsoft Research, One Microsoft Way, Redmond, WA 98052.  
Email: mahdian@microsoft.com.

## Abstract:

We study the behavior of algorithms for buffering packets weighted by different levels of Quality of Service (QoS) guarantees in a single queue. Buffer space is limited, and packet loss occurs when the buffer overflows. We describe a modification of the previously proposed "preemptive greedy" algorithm of Kesselman et al. for buffer management and give an analysis to show that this algorithm achieves a competitive ratio of at most 1.75. This improves upon recent work showing a 1.98 competitive ratio, and a previous result that shows a simple greedy algorithm has a competitive ratio of 2.

## **A Practitioner's Approach To Distribution And Supply Chain Planning And Management**

Tan Miller  
Senior Director Logistics  
Pfizer  
973 385 2619  
tan.miller@pfizer.com

The discussion will focus on some of the key components required to facilitate effective planning and management of both large-scale and small-scale supply chains. The discussion will begin with a review of general frameworks for organizing a firm's supply chain activities. These frameworks illustrate several keys of effective distribution and supply chain planning including:

- How to integrate short-run and long-run planning and scheduling activities over the entire global supply chain of a firm
- Methods for assuring that the activities of individual functional areas of a firm's supply chain (e.g., Distribution) are effectively integrated with the entire supply chain
- Methods for organizing and aligning a firm's performance metrics that monitor and evaluate the efficiency and effectiveness of supply chain operations

We will illustrate these frameworks with specific examples of implementations of these methodologies. These approaches are applicable to a broad spectrum of supply chain sub-processes and activities including production, distribution, forecasting, inventory management, transportation and so on, as well the total supply chain itself.



# Job Shop Scheduling with Setup Times, Deadlines and Precedence Constraints

Egon Balas\*

Carnegie Mellon University, Pittsburgh PA

Neil Simonetti

Bryn Athyn College of the New Church, Bryn Athyn PA

Alkis Vazacopoulos

Dash Optimization, Englewood Cliffs, NJ

January 2005

## Abstract

In the last 15 years several procedures have been developed that can find solutions of acceptable quality in reasonable computing time to Job Shop Scheduling problems in environments that do not involve sequence-dependent setup times of the machines. The presence of the latter, however, changes the picture dramatically. In this paper we adapt one of the best known heuristics, the Shifting Bottleneck Procedure, to the case when sequence dependent setup times play an important role. This is done by treating the single machine scheduling problems that arise in the process as Traveling Salesman Problems with time windows, and solving the latter by an efficient dynamic programming algorithm. The model treated here also incorporates precedence constraints, release times and deadlines. Computational experience on a vast array of instances, mainly from the semiconductor industry, shows our procedure to advance substantially the state of the art.

## 1 Introduction and Problem Description

In several manufacturing environments, scheduling is considered to be one of their most crucial components. By efficiently allocating specific resources to activities over time, organizations can maximize utilization of these resources, improve productivity, and reduce overall cost. Several scheduling applications can be found in the automobile, chemical, computer, semiconductor, printing, and pharmaceutical industries. Because of this, research in scheduling has grown dramatically in the last fifty years.

---

\*Research supported by the National Science Foundation through grant DMI-9802773 and by the Office of Naval Research through contract #N00014-97-1-0133

In this paper, we deal with a variant of the Job Shop Scheduling Problem. Release dates, deadlines, and sequence-dependent setup times are introduced. This latter feature significantly changes the nature of the problem and requires a new approach. The variant of the Job Shop Scheduling problem discussed here is a very common problem in the semiconductor industry [15].

In the job shop scheduling problem, job (items) are to be processed on machines (resources) while minimizing some function of completion times of the jobs, subject to constraints: (i) the sequence of machines for each job is prescribed, and (ii) each machine can process at most one job at a time. The processing of a job on a machine is called an operation, and its duration is a given constant. A time may be needed to adjust a machine between two consecutive operations (jobs), which is called a setup time, and which may or may not be sequence-dependent. Jobs may also have a release time and/or a deadline. The objective here is to minimize the time needed to complete all the jobs, also known as the makespan.

Let  $N = \{0, 1, \dots, n\}$  denote the set of operations, with 0 and  $n$  the dummy “start” and “finish” operations, respectively,  $M$  the set of machines,  $A$  the set of pairs of operations constrained by precedence relations representing condition (i) above, and  $E_k$  the set of pairs of operations to be performed on machine  $k$  and which therefore cannot overlap in time, as specified in (ii). Further, let  $c_{ij}$  denote the setup time required between every pair of operations (define  $c_{ij} := 0$  if  $(i, j) \notin E_k$  for any  $k$ ), let  $d_j$  denote the (fixed) duration or processing time, and let  $t_j$  be the (variable) start time of operation  $j$ . Finally, let  $L_i$  be the release time and  $U_i$  the deadline of the job associated with operation  $i$ , and let  $R$  and  $D$  be the sets of operations for which release times and deadlines are defined. The problem can then be stated as:

$$\begin{array}{ll}
\text{minimize} & t_n \\
\text{subject to} & \\
& t_j - t_i \geq d_i \quad (i, j) \in A \\
& t_i \geq 0 \quad i \in N \\
& t_j - t_i \geq d_i + c_{ij} \vee t_i - t_j \geq d_j + c_{ji} \quad (i, j) \in E_k, k \in M \\
& t_i \geq L_i, \quad i \in R \\
& t_i \leq U_i - d_i, \quad i \in D
\end{array} \tag{P}$$

Any feasible solution to (P) is called a schedule.

It is useful to represent this problem on a *disjunctive graph*  $G := (N, A, E)$  with node set  $N$ , (directed) arc set  $A$ , and (undirected, but orientable) edge set  $E$ . The length of an arc  $(i, j) \in A$  is  $d_i$ , whereas the length of an edge  $(i, j) \in E$  is either  $d_i + c_{ij}$  or  $d_j + c_{ji}$  depending on its orientation. Each machine  $k$  corresponds to a set  $N_k$  of nodes (operations) and a set  $E_k$  of edges which together form a disjunctive clique.

For the last fifteen years, job shop scheduling problems without sequence dependent setup times have been successfully treated with the Shifting Bottleneck Procedure described in [1], improved upon in [4, 6], and extended to the case of jobs with deadlines in [3]. This is a heuristic procedure without a theoretical performance guarantee but with a solid track record of empirical performance. Its driving engine is an efficient procedure used repeatedly, for solving a certain type of one-machine problem. However, the introduction of sequence-dependent setup times changes the nature of the one-machine problems and thereby invalidates the earlier approach. The current paper’s main contribution is to treat the one-machine problem

with release times, deadlines and sequence-dependent setup times as a Traveling Salesman Problem (TSP) with time windows, and solve it by a dynamic programming algorithm designed for this purpose. The efficiency of the procedure is demonstrated on a large array of instances from the literature, mainly from the semiconductor industry.

Our paper is organized as follows: Section 2 describes the Shifting Bottleneck Procedure and explains the need for a new approach to solve its one-machine problems in the presence of sequence-dependent setup times. It then formulates this problem as an asymmetric TSP with time windows determined by the release times and due dates resulting from the dynamics of the evolving overall schedule, along with the fixed deadlines. section 3 reduces this problem to an asymmetric TSP with a special type of precedence constraints solvable by a dynamic programming algorithm whose complexity is linear in the number of operations. Finally, section 4 discusses our computational experience and comparisons with other approaches.

## 2 The Shifting Bottleneck and One-Machine Problem

The Shifting Bottleneck Procedure [1] sequences the machines consecutively, one at a time, with the remaining unsequenced machines ignored (i.e. the corresponding edge sets removed) and the machines already sequenced held fixed (i.e. the corresponding edge sets replaced by directed arcs). At each step a bottleneck machine is determined from among those not yet sequenced, by solving a one machine scheduling problem for each unsequenced machine and choosing the one with maximum makespan. This bottleneck machine is then sequenced optimally by using the solution to the corresponding one-machine problem. After a machine has been sequenced, each machine in turn is freed up and resequenced, with the sequences on the remaining machines held fixed. Thus a concise statement of the procedure is as follows:

Let  $M_0$  be the set of machines already sequenced ( $M_0 = \emptyset$  at the start).

**Step 1.** Identify a bottleneck machine  $m$  among the machines  $k \in M \setminus M_0$  and sequence it optimally. Set  $M_0 \leftarrow M_0 \cup \{m\}$  and go to 2.

**Step 2.** Reoptimize the current partial schedule. If  $M_0 = M$ , stop; otherwise go to 1.

An improved version of the SB procedure, developed in [6], replaces Step 2 by a guided local search based on an interchange scheme and using the concept of neighborhood trees. Structural properties of the neighborhood are used to guide the search in promising directions. While the resulting procedure, SB-GLS, is computationally more expensive, the increase in cost is compensated by a significant improvement in the quality of solutions. Further improvements in solution quality can be obtained at a steeper computational cost by randomizing the choice of machines to be resequenced in step 2, and repeating this step  $h$  times for some positive integer  $h$ . This variant is called SB-RGLSh (see [6] for details).

Obviously, the success of the Shifting Bottleneck (SB) approach depends primarily on our ability to solve efficiently the one-machine problems that arise during the procedure. These problems can be described as follows. At a given stage of the SB procedure, let  $M_0$  be the set of machines already sequenced, and let  $G(M_0)$  be the disjunctive graph obtained from  $G$  by replacing all disjunctive arc sets corresponding to the machines in  $M_0$  by the conjunctive arc sets representing the sequences chosen for those machines, and deleting all disjunctive arc sets corresponding to the machines in  $M \setminus (M_0 \cup \{k\})$  for some fixed  $k \in M \setminus M_0$ . Then

the one-machine problem to be solved for machine  $k$  can be stated as

$$\begin{array}{llll}
\text{minimize} & t_m & & \\
\text{subject to} & & & \\
& t_m - t_i \geq d_i + q_i, & i \in I & \\
& t_i \geq r_i, & i \in I & \\
& t_i \leq U_i - d_i - q_i & i \in I \cap D & \\
t_j - t_i \geq d_i + c_{ij} \quad \forall & t_i - t_j \geq d_j + c_{ji} & (i, j) \in E_k, & P(k, M_0)
\end{array}$$

where  $I$  is the set of jobs to be processed on machine  $k$ ,  $m = |I|$ ,  $r_i$  is the release time (or head) of job  $i$ , equal to the length  $L(0, i)$  of the longest path from node 0 to node  $i$  in  $G(M_0)$ , and  $q_i$  is the delivery time (or tail) of job  $i$ , defined as  $L(i, n) - d_i$ , where  $L(i, n)$  is the length of a longest path from node  $i$  to node  $n$  in  $G(M_0)$ .

Although this problem is  $\mathcal{NP}$ -complete in the strong sense, in the absence of sequence-dependent setup times and deadlines it has a very nice feature that makes it efficiently solvable in most cases. This feature consists in the fact that if  $r_i = r_j$  for all  $i, j \in I$ , then the Longest Tail rule, which schedules the jobs in order of decreasing tail length, yields an optimal schedule. Furthermore, when the schedule produced by this rule is not optimal, then a set of jobs  $J \subset I$  and a job  $p \in I \setminus J$  are readily determined, such that in any feasible schedule  $p$  comes either before, or after all  $j \in J$ . This leads to a highly efficient branching rule (see Carlier [10]) that permits the solution of  $P(k, M_0)$  with search trees typically not exceeding  $O(|I|)$  nodes.

Unfortunately, in the presence of sequence-dependent setup times, this nice feature of the problem vanishes, and therefore a different approach is needed if  $P(k, M_0)$  is to be solved efficiently. Notice that  $P(k, M_0)$  is essentially the problem of ordering  $I$  into an optimal sequence, given upper and lower bounds on the position of each item in the sequence, as well as a minimum time-distance between each pair of items. In other words,  $P(k, M_0)$  can be viewed as a TSP with time windows. This, of course, is also true in the absence of sequence-dependent setup times; but in that case the TSP feature is not significant. To see this, it suffices to point out that if the cost of traveling from city  $i$  to any other city is  $d_i$ , then the cost of a TSP tour is  $\sum(d_i : i \in I)$ , irrespective of the order in which the cities are visited. If there are setup times but they are not sequence-dependent, more precisely they depend only on one job instead of a pair, then they can be incorporated into the processing times  $d_i$  and the problem remains essentially unchanged. It is the presence of sequence-dependent setup costs that makes the TSP feature significant, and makes it worth stating the problem as an asymmetric TSP with time windows.

### 3 The TSP with Precedence Constraints

In the variant of the TSP with time windows (TSPTW) relevant to our case, we are given a set  $I$  of cities, a travel time  $t_{ij}$  between cities  $i$  and  $j$  for all pairs  $i, j \in I$ , and a time window  $[a_i, b_i]$  for each city  $i$ , with the interpretation that city  $i$  has to be visited not earlier than  $a_i$  and not later than  $b_i$ . If city  $i$  is reached before  $a_i$ , there is a waiting time  $w_i$  until  $a_i$ ; but if city  $i$  is visited later than  $b_i$ , the tour is infeasible. The objective is to find a feasible tour of minimum total duration, including the waiting times. In the case of the

one-machine problem  $P(k, M_0)$  stated as a TSPTW,  $t_{ij} = d_i + c_{ij}$ ,  $i, j \in I$ ,  $a_i = r_i$ , and  $b_i = \min\{t_m^*, U_i\} - d_i - q_i$  for all  $i \in I$ , where  $t_m^*$  is the length of a feasible schedule. Since the upper bounds of all the time windows depend on the value of  $t_m^*$ , the problem in fact asks for the smallest value  $t_m^{**}$  of  $t_m^*$ , for which a feasible tour exists, as well as for an optimal tour given that  $t_m^* = t_m^{**}$ . Here  $t_m^{**}$  can be determined by binary search or some other procedure that solves the problem for different values of  $t_m^*$ .

The TSPTW in turn can be reformulated as a TSP with precedence constraints of a special type, for which an efficient dynamic programming algorithm has been proposed in [2] and implemented in [5]. This algorithm has the property of searching an exponentially large neighborhood in time linear in the number of cities.

To be more specific, let  $I$  be a set of cities and  $t_{ij}$  the travel time between  $i$  and  $j$ , for  $i, j \in I$ . Given a positive integer  $k$  and an initial ordering  $\sigma = \{1, \dots, m\}$  of  $I$ , we are asked to find an optimal tour, i.e. permutation  $\pi$  of  $\sigma$ , satisfying  $\pi(1) = 1$  and

$$\pi(i) < \pi(j) \text{ for all } i, j \text{ such that } i + k \leq j. \quad (1)$$

In other words, if city  $i$  precedes city  $j$  by  $k$  or more places in the initial ordering, then city  $i$  must be visited before city  $j$  in an optimal tour.

TSP's that satisfy condition (1) can be solved by dynamic programming in time linear in  $m = |I|$  (though exponential in  $k$ ). Furthermore, this property still holds if  $k$  is replaced by city-specific constants  $k(i) < 0$ ,  $i \in I$ . The algorithm in question constructs an auxiliary network  $G^*$  with a source, a sink, and a layer of nodes for each position in the tour, such that there is a 1-1 correspondence between tours satisfying condition (1) and source-sink paths in  $G^*$ . An optimal tour satisfying (1) is then found by constructing a shortest source-sink path in  $G^*$ . Since the size of the layers is exponential in  $k$ , memory is a limiting factor in this approach. At the current state of the art, values of  $k$  beyond 20 are not practical.

One of the nice features of this approach is that when a TSP does *not* satisfy condition (1) because the constant  $k$  is not large enough (or we do not wish to invest the time required to run the algorithm for a large value of  $k$ ), then the dynamic programming algorithm, no longer guaranteed to find an optimal solution, can still be used as a linear-time heuristic that examines an exponential-sized neighborhood.[5]

Next we discuss how the TSPTW can be formulated as a TSP with condition (1). Given an initial ordering of the cities, the time windows can be used to derive precedence constraints of the type desired. Indeed, in any feasible tour, city  $i$  has to precede any city  $j$  such that  $a_i + t_{ij} > b_i$ . If  $j_0$  is the smallest index such that  $a_j + t_{ji} > b_i$  for all  $j \geq j_0$ , then we can define  $k(i) = j_0 - i$ . Doing this for every  $i$ , we obtain a TSP with condition (1) whose feasible solutions include all feasible tours for the TSP with the time windows  $[a_i, b_i]$ . Since the converse is not necessarily true, the dynamic programming routine has to be amended with a feasibility check: while traversing  $G^*$ , paths that violate the upper limit of some time window must be weeded out, whereas paths that violate a lower limit of some time window must have their lengths (costs) increased by the waiting time. This check is a simple comparison that does not affect the complexity of the algorithm.

Clearly, the size of the constants  $k(i)$ ,  $i \in I$ , derived in this way depends on the tightness of the time windows as well as on the quality ("distance" from the optimum) of the initial ordering. One way of choosing the latter is to sort the time windows by their midpoint. Typically this gives a reasonable-sized largest  $k(i)$ , which can often be improved (reduced) by heuristically rearranging (interchanging) some of the time windows.

As the TSPTW formulation of the one-machine problem has to be solved repeatedly for different values of  $t_m^*$ , the time windows themselves have to be redefined for each new  $t_m^*$ : to be more specific, as in practice we solve the problem for a sequence of diminishing values of  $t_m^*$  (see below), the time windows get tightened after each such iteration.

If the time-windows of the TSPTW problem imply precedence constraints of type (1) with a value of  $k := \text{mboxmax}_i k(i)$  larger than is practical to use, then the exponential neighborhood search over the neighborhood defined by (1) with the largest affordable  $k$  serves as a heuristic, which still performs very well on TSPTW instances [5].

As a TSP tour is built by dynamic programming, the states in the dynamic program for the TSP [13] corresponding to paths specify the last node in the path and the set of nodes already visited in the path. A simple test at each node can enforce additional precedence constraints generated by the scheduling problem. Furthermore, because of the nature of constraint (1) we can be certain that any path of length  $l$  corresponding to a state in the above dynamic program will have visited nodes 1 through  $l - k$ , and will not have visited nodes  $l + k$  through  $n$ . This allows for a constant-time test for precedence constraints, preserving the linear-time complexity of the algorithm.

The presence of delayed precedence constraints (see [5]) creates a hurdle that the dynamic program cannot handle entirely. Given a required delay of  $w_{ij}$  between the completion of operation  $i$  and the beginning of operation  $j$ , the cost of the arc from node  $i$  to node  $j$  in the corresponding TSP can be changed from  $c_{ij}$  to  $w_{ij}$  if  $w_{ij} > c_{ij}$ , but this cannot enforce the delayed constraint if additional jobs are scheduled between operations  $i$  and  $j$ . Still, the algorithm performs well as a heuristic to find the minimum target makespan  $M$ , which will be seen in section 4.

## 4 Computational Results

We have implemented our procedure in C and have run all our tests on a Sun Ultra 60 using a 360 MHz UltraSPARC-II processor. Complete results can be viewed at <http://www.andrew.cmu.edu/~neils/tsp>.

### 4.1 Description of Test Data

We have tested our algorithms primarily on the rich data sets generated by Ovacik and Uzsoy [14, 15]. These instances were inspired by the actual environment in the semiconductor industry. We briefly summarize their description by the authors.

The 960 instances of [14] are artificially constructed classical job shop scheduling instances with the addition of setup times. Each job has to be processed on each machine exactly once. The jobs visit the machines in a predetermined sequence and all machines have sequence-dependent setup times. The machine sequence for each job is a random permutation of the machines, while the processing times and setup times are uniformly distributed random numbers. Jobs also have deadlines, uniformly distributed on an interval determined by the expected workload of the system and several other parameters.

The 1920 instances of [15] come from a semiconductor testing facility consisting of a number of testing workcenters and a brand workcenter. This corresponds to the post-burn-in portion of an actual large testing facility. Processing and setup times were derived from

data gathered in the actual facility, otherwise the structure of the instances follows the pattern described in the previous paragraph.

In both cases, the instances are grouped into six classes, which differ by certain properties of the data. Within each class, the instances are grouped into sets of 20, where each set involves the same number of jobs and the same number of machines. For more details, see [14, 15]. Data for these problems can be found at:

<http://palette.ecn.purdue.edu/~uzsoy2/Problems/classic/nomenclature.html>

[http://palette.ecn.purdue.edu/~uzsoy2/Problems/semic/single\\_nomenclature.html](http://palette.ecn.purdue.edu/~uzsoy2/Problems/semic/single_nomenclature.html)

## 4.2 Adding a Guided Local Search

As mentioned in section 2, combining the Shifting Bottleneck with a Guided Local Search on standard job shop scheduling problems improves the quality of solutions at a computational cost. While in the standard case this extra cost is usually amply justified, it was open to question whether this is still true under the changed circumstances produced by the setup times, where the one machine problems are processed by the dynamic programming routine. Therefore our first experiments were directed at settling this issue. Table 1 compares the algorithm based on the standard Shifting Bottleneck procedure with one that uses the combined algorithm on the first of the six classes of instances in [14], the data set js305. As it can be seen, the combined algorithm obtains significantly better results, at the cost of what seems a tolerable increase in computing time.

Table 1: Results with and without local search.

| Average of Maximum Lateness (avg. CPU seconds) |      |          |                   |          |                      |         |
|------------------------------------------------|------|----------|-------------------|----------|----------------------|---------|
| Problems                                       | Jobs | Machines | with local search |          | without local search |         |
| 1-20                                           | 10   | 5        | 360.20            | (1.86)   | 421.95               | (0.31)  |
| 21-40                                          | 20   | 5        | 91.00             | (61.43)  | 150.75               | (14.39) |
| 41-60                                          | 10   | 10       | 1082.25           | (6.00)   | 1268.65              | (0.37)  |
| 61-80                                          | 20   | 10       | 803.45            | (103.53) | 1106.95              | (23.91) |
| 81-100                                         | 10   | 15       | 1658.10           | (14.12)  | 1907.05              | (0.51)  |
| 101-120                                        | 20   | 15       | 1531.45           | (114.30) | 1895.85              | (27.26) |
| 121-140                                        | 10   | 20       | 2166.05           | (22.74)  | 2508.90              | (0.92)  |
| 141-160                                        | 20   | 20       | 2070.50           | (109.88) | 2702.55              | (26.63) |
| Dataset js305, $k = 15$                        |      |          |                   |          |                      |         |

## 4.3 Step-Down versus Binary Search

As mentioned in section 3, when finding the optimal target makespan  $M$  for the one-machine scheduling problem, a binary search can be used. In practice, a “step-down” approach requires fewer than  $\log(M)$  steps to find this value of  $M$ , and the earlier steps can be taken more quickly by using smaller values for the parameter  $k$  mentioned in dynamic programming model outlined in section 2. Starting with a target makespan large enough to ensure

that any sequence is feasible (effectively infinity), the dynamic program will find, within its exponential neighborhood, the tour that minimizes the setup times. This will not minimize the makespan, but it will find a reasonable quality makespan of, say,  $M_1$ . The algorithm can now be run with a target makespan of  $M_1 - 1$  in an effort to improve on the makespan. If an improvement is found, it will have a makespan of  $M_2 = M_1 - m$  for some positive integer  $m$ , and the next target makespan would be  $M_2 - 1$ . The process is repeated until no improvement is found. Because  $M_1$  is usually not far from the optimal  $M$ , this approach takes fewer than  $\log(M)$  steps in practice.

Another advantage of this “step-down” approach is that a smaller value of  $k$  can be used for the first few steps of the process, when the windows are bigger and feasible solutions are easier to find.

Table 2 shows this advantage on a subset of the test problems used. The first column shows the results when  $k = 15$  is used for the “step-down” process, and the second column shows the results when  $k = 10$  is used until no feasible solution is found, and then  $k = 15$  is used to continue the “step-down” process. The running times for the second column are significantly lower, while the results are similar. The reason the results may differ is that the procedure is still only a heuristic, and a different sequence of “step-downs” may cause the algorithm to search different neighborhoods. Also, one different solution to a one-machine scheduling problem may create a different path of optimizations by the shifting bottleneck.

## 4.4 Comparisons with Ovacik-Uzsoy for different values of $k$

In table 3, we compare the performance of our procedure to that of Ovacik and Uzsoy [14, 15], based on a variety of dispatching rules. We also compare the performance of our procedure with two different values of the parameter  $k$  used in the dynamic programming routine to solve the one-machine problems. In both cases,  $k = 10$  is used for the first part of the “step-down”. The “step-down” is then continued with either  $k = 12$ , or  $k = 15$ . A summary of the results on two of the twelve data sets is shown in this table, which is typical of all the results. For full results, see <http://www.andrew.cmu.edu/~neils/tsp>.

Of the 960 problems introduced in [14], 480 have 20 jobs (problems with 10 jobs give the same results for any value of  $k$  larger than or equal to 10). Of these 480 problems, 90 (18.8%) had better solutions with  $k = 12$ , 92 (19.2%) had better solutions with  $k = 15$ , and 298 (62.1%) had the same solution with both  $k$  values. Comparing to Ovacik and Uzsoy, both  $k = 12$  and  $k = 15$  outperformed [14] in all but three of the 960 instances (99.4%).

Of the 1920 semiconductor problems introduced in [15], 1440 have more than 10 jobs. Of these 1440 problems, 170 (11.8%) had better solutions with  $k = 12$ , 223 (15.5%) had better solutions with  $k = 15$ , and 1047 (72.7%) had the same solution with both  $k$  values. In the larger problems from this set, running times for  $k = 15$  were up to 10 times longer than  $k = 12$  in some cases.

Comparing to Ovacik and Uzsoy, both  $k = 12$  and  $k = 15$  outperformed [15] in 1862 of the 1920 instances (97.0%).

## 4.5 Other Comparisons

We have also tested our algorithms on a set of problems from Brucker and Thiele [8]. These are jobshop scheduling problems with setup times derived from the corresponding benchmark



Table 2: Illustrating a benefit of the “step-down” search

| <i>Problem Set i305</i> |      |          | Maximum Lateness (CPU seconds) |                           |
|-------------------------|------|----------|--------------------------------|---------------------------|
| Problem Number          | Jobs | Machines | step-down<br>$k = 15$          | step-down<br>$k = 10, 15$ |
| 21                      | 20   | 5        | 625 (212.92)                   | 521 (55.16)               |
| 22                      | 20   | 5        | -40 (172.07)                   | -158 (50.68)              |
| 23                      | 20   | 5        | 161 (138.14)                   | 156 (47.23)               |
| 24                      | 20   | 5        | 5 (172.51)                     | 5 (65.61)                 |
| 25                      | 20   | 5        | 7 (217.05)                     | -64 (54.07)               |
| 61                      | 20   | 10       | 649 (295.07)                   | 649 (102.46)              |
| 62                      | 20   | 10       | 803 (415.40)                   | 698 (156.15)              |
| 63                      | 20   | 10       | 1095 (283.24)                  | 1095 (110.98)             |
| 64                      | 20   | 10       | 988 (348.94)                   | 988 (109.41)              |
| 65                      | 20   | 10       | 761 (381.15)                   | 593 (107.38)              |
| 101                     | 20   | 15       | 1701 (367.58)                  | 1701 (107.49)             |
| 102                     | 20   | 15       | 1784 (390.45)                  | 1530 (108.76)             |
| 103                     | 20   | 15       | 1453 (343.92)                  | 1606 (110.93)             |
| 104                     | 20   | 15       | 1934 (338.95)                  | 1934 (130.99)             |
| 105                     | 20   | 15       | 1756 (462.22)                  | 1766 (145.28)             |
| 141                     | 20   | 20       | 1871 (285.09)                  | 1871 (110.47)             |
| 142                     | 20   | 20       | 1902 (296.88)                  | 1902 (98.71)              |
| 143                     | 20   | 20       | 2223 (261.57)                  | 2223 (92.50)              |
| 144                     | 20   | 20       | 2181 (272.23)                  | 2058 (109.08)             |
| 145                     | 20   | 20       | 2074 (306.18)                  | 1929 (103.30)             |

Table 3: Summary of results on job-shop scheduling problems.

| Average percent improvement of total<br>makespan vs. Ovacik and Uzsoy (avg. CPU seconds)                                                  |      |          |             |          |             |           |
|-------------------------------------------------------------------------------------------------------------------------------------------|------|----------|-------------|----------|-------------|-----------|
| Problems                                                                                                                                  | Jobs | Machines | $k = 10/12$ |          | $k = 10/15$ |           |
| Dataset js305 [14]                                                                                                                        |      |          |             |          |             |           |
| 1-20                                                                                                                                      | 10   | 5        | 16.25       | (1.88)   | 16.25       | (1.86)    |
| 21-40                                                                                                                                     | 20   | 5        | 16.85       | (16.42)  | 17.30       | (61.43)   |
| 41-60                                                                                                                                     | 10   | 10       | 24.90       | (6.07)   | 24.90       | (6.00)    |
| 61-80                                                                                                                                     | 20   | 10       | 21.95       | (42.05)  | 22.20       | (103.53)  |
| 81-100                                                                                                                                    | 10   | 15       | 27.90       | (14.15)  | 27.90       | (14.12)   |
| 101-120                                                                                                                                   | 20   | 15       | 24.30       | (75.04)  | 24.25       | (114.30)  |
| 121-140                                                                                                                                   | 10   | 20       | 31.65       | (23.09)  | 31.65       | (22.74)   |
| 141-160                                                                                                                                   | 20   | 20       | 26.85       | (97.52)  | 27.05       | (109.88)  |
| Dataset i305 [15]                                                                                                                         |      |          |             |          |             |           |
| 1-20                                                                                                                                      | 10   | 6        | 7.10        | (0.50)   | 7.05        | (1.25)    |
| 21-40                                                                                                                                     | 10   | 11       | 14.05       | (0.64)   | 14.05       | (0.72)    |
| 41-60                                                                                                                                     | 10   | 16       | 18.45       | (0.57)   | 18.45       | (0.57)    |
| 61-80                                                                                                                                     | 10   | 21       | 17.05       | (0.47)   | 17.05       | (0.46)    |
| 81-100                                                                                                                                    | 25   | 6        | 11.55       | (5.95)   | 11.70       | (29.82)   |
| 101-120                                                                                                                                   | 25   | 11       | 15.50       | (5.41)   | 15.50       | (29.99)   |
| 121-140                                                                                                                                   | 25   | 16       | 15.85       | (3.81)   | 15.90       | (18.65)   |
| 141-160                                                                                                                                   | 25   | 21       | 18.10       | (11.00)  | 18.10       | (68.46)   |
| 161-180                                                                                                                                   | 50   | 6        | 9.85        | (34.63)  | 9.80        | (299.95)  |
| 181-200                                                                                                                                   | 50   | 11       | 13.40       | (49.70)  | 13.25       | (421.52)  |
| 201-220                                                                                                                                   | 50   | 16       | 12.80       | (47.18)  | 12.85       | (390.62)  |
| 221-240                                                                                                                                   | 50   | 21       | 14.40       | (66.48)  | 14.30       | (457.56)  |
| 241-260                                                                                                                                   | 75   | 6        | 9.75        | (74.22)  | 9.90        | (585.99)  |
| 261-280                                                                                                                                   | 75   | 11       | 12.20       | (109.08) | 12.25       | (876.68)  |
| 281-300                                                                                                                                   | 75   | 16       | 11.25       | (157.13) | 11.35       | (1112.69) |
| 301-320                                                                                                                                   | 75   | 21       | 12.40       | (245.68) | 12.40       | (1152.96) |
| Results for all 12 data sets can be viewed at:<br><a href="http://www.andrew.cmu.edu/~neils/tsp">http://www.andrew.cmu.edu/~neils/tsp</a> |      |          |             |          |             |           |

problems tested in [1]. Brucker and Thiele have introduced groups according to the format specification, whereby each operation of the same job belongs to the same group, and there are sequence dependent setup times between operations from different groups.

Instances ps01–ps05, ps06–ps10, and ps11–ps15 are  $10 \times 5$  (ten jobs, five machines),  $15 \times 5$ , and  $20 \times 5$ , respectively; and instances pss12 and pss13 are also  $20 \times 5$ . For more details, see [8]. The last two instances were added to the set of 15 because they had been solved by Focacci et al. [12] and by Buscaylet and Artigues [9].

Table 4 compares our solutions with the results from Brucker and Thiele [8], Focacci, Laborie and Nuijten [12], and Buscaylet and Artigues [9]. Brucker and Thiele’s algorithm is a specialized branch and bound procedure. Focacci et al. use constraint programming with lower bounds obtained by linear programming. For the larger problems, their procedure has two phases, each allowed to run for 60 seconds. The first is aimed at minimizing the makespan, the second at minimizing total setup time with the makespan bounded by the value found in the first phase. For the smaller problems, ps01–ps05, Focacci et al. would allow the first phase to run until the optimal makespan was found (to a limit of 30 minutes). The algorithm of Buscaylet and Artigues is a tabu search performed on the best schedule from 1000 schedules generated from a strict order schedule generation scheme with a random priority rule.

As to our approach, besides the algorithm SB-GLS described in Section 2, we also used a more computation-intensive version shown in the last column of Table 4 as Multi-Run SB-RGLS10. This is a two-phase procedure whose first phase is the algorithm SB-RGLS10 outlined in Section 2, and whose second phase is a modified version of SB-RGLS10 in which the bottleneck machine is chosen at random from a uniform distribution; the second phase is repeated 40 times. This procedure is available at:

<http://www.andrew.cmu.edu/~neils/tsp>.

Comparing the data of Table 4 for the different procedures does not easily lend itself to conclusions, partly because of the incompleteness of some of the runs, partly because of the differences in the computers used. It is clear, however, that our approach compares much less favorably on the smaller  $10 \times 5$  instances (ps01–ps05), than on the harder  $15 \times 5$  and  $20 \times 5$  instances. On the hard instances, our procedures are way ahead of all the others in terms of solution quality, and also better in terms of computing times. One possible reason for this is that the role of setup times is increased in the larger instances, since the single-machine problems become larger. The number of distinct setup times per machine grows with the square of the number of jobs, and the time needed to solve the associated TSPTW increases exponentially for most algorithms. Since the running time of our TSPTW algorithm increases linearly with the number of jobs (once  $k$  is selected), it is not surprising that our procedure does better on the instances with more jobs per machine and the results testify to the suitability of our approach to dealing with sequence-dependent setup times.

## References

- [1] Adams, J., Balas, E. and Zawack, D., 1988. The Shifting Bottleneck Procedure for Job-Shop Scheduling. *Management Science*, 34, 391-401.

Table 4: Comparison with a Randomized Shifting Bottleneck

| CPU times in parentheses                                                                         |                         |                                 | [9]        | Single-Run<br>SB-GLS | Multi-Run<br>SB-RGLS10 |
|--------------------------------------------------------------------------------------------------|-------------------------|---------------------------------|------------|----------------------|------------------------|
| Instance                                                                                         | [8]                     | [12]                            |            |                      |                        |
| t2-ps01                                                                                          | 798 <sup>1</sup> (502)  | 798(1800)                       | 800(14.0)  | 815(1.92)            | 798(358)               |
| t2-ps02                                                                                          | 784 <sup>1</sup> (158)  | 784 <sup>1</sup> (88)           | 784(85.1)  | 807(1.70)            | 784(550)               |
| t2-ps03                                                                                          | 749 <sup>1</sup> (1892) | 749 <sup>1</sup> (144)          | 764(103)   | 771(1.76)            | 749(834)               |
| t2-ps04                                                                                          | 730 <sup>1</sup> (189)  | 730 <sup>1</sup> (388)          | 745(209)   | 738(1.46)            | 730(515)               |
| t2-ps05                                                                                          | 691 <sup>1</sup> (770)  | 691 <sup>1</sup> (30)           | 703(216)   | 693(1.21)            | 693(248)               |
| t2-ps06                                                                                          | 1056(7200)              | -                               | -          | 1027(11.0)           | 1018(1192)             |
| t2-ps07                                                                                          | 1087(7200)              | -                               | -          | 1022(11.1)           | 1003(1338)             |
| t2-ps08                                                                                          | 1096(7200)              | -                               | -          | 1013(10.1)           | 975(833)               |
| t2-ps09                                                                                          | 1119(7200)              | -                               | -          | 1101(8.31)           | 1060(423)              |
| t2-ps10                                                                                          | 1058(7200)              | -                               | -          | 1051(9.07)           | 1018(557)              |
| t2-ps11                                                                                          | 1658(7200)              | -                               | -          | 1572(49.9)           | 1470(3047)             |
| t2-ps12                                                                                          | 1528(7200)              | 1450(60)/1448(120) <sup>2</sup> | 1464(178)  | 1369(40.8)           | 1305(2173)             |
| t2-ps13                                                                                          | 1549(7200)              | 1669(60)/1658(120) <sup>2</sup> | 1575(149)  | 1439(67.5)           | 1439(2468)             |
| t2-ps14                                                                                          | 1592(7200)              | -                               | -          | 1602(43.3)           | 1485(2131)             |
| t2-ps15                                                                                          | 1744(7200)              | -                               | -          | 1542(51.4)           | 1527(3111)             |
| t2-pss12                                                                                         | 1384(7200)              | 1367(60)/1362(120) <sup>2</sup> | 1359(85.3) | 1305(48.9)           | 1290(2062)             |
| t2-pss13                                                                                         | 1463(7200)              | 1522(60)/1518(120) <sup>2</sup> | 1470(126)  | 1409(54.6)           | 1398(1875)             |
| Results in [8] used a Sun 4/20 workstation (20MHz)                                               |                         |                                 |            |                      |                        |
| <sup>1</sup> These results have a guarantee of optimality.                                       |                         |                                 |            |                      |                        |
| Results in [12] used a Pentium II (300MHz)                                                       |                         |                                 |            |                      |                        |
| <sup>2</sup> Results shown here are from each phase of a two phase procedure.                    |                         |                                 |            |                      |                        |
| Results in [9] used a PC (2.8MHz)                                                                |                         |                                 |            |                      |                        |
| Our results used a Sun Ultra 60 (360Mhz),<br>$k = 10/15$ for the dynamic programming subroutine. |                         |                                 |            |                      |                        |

- [2] Balas, E., 1996. New Classes of Efficiently Solvable Generalized Traveling Salesman Problems, Management Science Research Report #MSRR-615, Graduate School of Industrial Administration, Carnegie Mellon University.
- [3] Balas, E., Lancia, G., Serafini, P. and Vazacopoulos, A., 1998. Job Shop Scheduling with Deadlines. *Journal of Combinatorial Optimization*, 1, 329-353.
- [4] Balas, E., Lenstra, J.K. and Vazacopoulos, A., 1995. The One-Machine Problem with Delayed Precedence Constraints and its Use in Job Shop Scheduling. *Management Science*, 41, 94-109.
- [5] Balas, E. and Simonetti, N., 2001. Linear Time Dynamic Programming Algorithms for New Classes of Restricted TSPs. *Inform Journal on Computing*, 13, 56-75.
- [6] Balas, E. and Vazacopoulos, A., 1998. Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. *Management Science*, 44, 262-275.
- [7] Baptiste, P., C. Le Pape, W. Nuijten. 2001. *Constrained-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers.
- [8] Brucker, P. and Thiele, O., 1996. A Branch and Bound Method for the General-Shop Scheduling Problem with Sequence Dependent Setup Times, *Operations Research Spektrum*, 18, 145-161.
- [9] Buscaylet, F. and Artigues, C., 2003. A Fast Tabu Search Method for the Job-Shop Problem with Sequence-Dependent Setup Times, MIC 2003, The Fifth Metaheuristics International Conference, Kyoto, Japan.
- [10] Carlier, J., 1982. The One-Machine Sequencing Problem. *European Journal of Operational Research*, 11, 42-47.
- [11] Carlier, J. and Pinson, E., 1989. An Algorithm for Solving the Job Shop Problem. *Management Science*, 35, 164-176.
- [12] Focacci, F., Laborie, P., and Nuijten, W., 2000. Solving Scheduling Problems with Setup Times and Alternative Resources. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenbridge, Colorado, 92-101.
- [13] Held, M. and Karp, R.M., 1962. A Dynamic Programming Approach to Sequencing Problems. *SIAM Journal of Applied Mathematics*, 10, 196-210.
- [14] Ovacik, I. and Uzsoy, R., 1994. Exploiting Shop Floor Status Information to Schedule Complex Job Shops. *Journal of Manufacturing Systems*, 13(2), 73-82.
- [15] Ovacik, I. and Uzsoy, R., 1997. Decomposition Methods for Complex Factory Scheduling Problems, Kluwer Academic Publishers, 1997.

# Worst-case Analysis for Queueing Service Policies \*

Zhenbo Wang, Wenxun Xing <sup>†</sup>

Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China

Nowadays in China, many banks have developed queueing service systems for their customers. When a customer arrives, he inputs his service demand into the machine and picks up a number, which identifies his arriving order and required service. A bank offers many normal services like opening a new account, deposit, withdrawal, and some special services like declaring the loss of bank cards, foreign money exchange; etc. To declare the loss of a bank card or an account book, or to exchange foreign currency usually needs special services, thus many banks open special windows for them. Usually the special windows can give normal and special services, but the normal windows can only do normal services. For simplicity, we call the windows offering special services as special windows and customers needing special services as special customers, and the same for normal windows and normal customers.

This scenario usually can be found in many service or manufacturing systems like web service demands, airplane ticket checking, electronic equipment repair, vehicle repair, product processing; etc.

Usually the number of special windows is predetermined by the managers of the service system according to the statistics of demands and the workload of windows for a long period, the facility costs, etc. For example, the number of foreign money exchange windows is determined by the demand of the foreign money exchange affairs for a long period and the number of trained cashiers.

In these systems, many service policies are adopted by their managers. The first and the easiest one is the ‘first come, first serve’(FCFS). All customers are in one line according to their arriving order and interruption of the order is not allowed.

If the special customers are considered more important than the normal customers, or there are excessive special customers, the ‘special windows serve special customers’ (SWSC) policy can be adopted. That is to say, there are two lists: ‘normal service’ and ‘special service’ lists, and the special windows process only special services.

If both impartiality and system optimization are considered, a third policy, the ‘first fit, first serve’ (FFFS) policy, may be adopted. This policy first follows the ‘first come, first serve’ rule, and then always lets the first normal customer waiting in line be served on an idle window if all special windows are busy. In this case, a normal customer is served even if there are special customers waiting in line before this customer when a normal window is idle. This policy fully utilizes all windows.

Synthesizing the last two service policies, we get a hybrid service window policy (HSW): some special windows serve according to ‘special windows for special customers’ policy, and some special windows serve according to ‘first fit, first serve’ policy.

Whenever in an emergency case or during a rush period, there are customers waiting in a long line for services. Customers can be considered as they come at the same time and make up of a list according to their arriving order. The managers of the queueing systems care about how to serve the customers as expeditiously as possible. This provides the motivation for us to use the worst-case analysis method to evaluate the performance of different service policies.

If the customers in our problem are considered as jobs, the windows are considered as machines, this queueing service problem is a parallel machine scheduling problem. The  $m$  machines are divided into two classes, special and normal machines. The jobs are divided into two classes as well, special and normal jobs. A normal job can be processed on both special and normal machines with the same speed. A

---

\*This paper was partially supported by Grant 9732003-12-124 and NSFC 70471008.

<sup>†</sup>Corresponding author. Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China. Email: wxing@math.tsinghua.edu.cn

special job can be processed only on special machines; it is not allowed to be processed on any normal machine. Our problem is a general case of the parallel machine scheduling problem  $Pm | C_{\max}$  and a special case of unrelated parallel machine scheduling problem  $Rm | C_{\max}$ . Preemption is not allowed in our discussion. As it is a scheduling problem, we naturally use algorithms to represent the four service policies and then give their worst-case ratios by the worst-case analysis method and skills of scheduling.

All the worst case performance ratios of the policies are calculated using their makespan  $C_{\max}$  relative to  $C_{\max}^*$ , where  $C_{\max}^*$  is assumed to be the off-line optimal value of our problem. For a service system with  $m$  windows, denote the tightly upper bounds of the worst-case ratios for policies FCFS, SWSC and FFFS as  $R_{FCFS}(k, m)$ ,  $R_{SWSC}(k, m)$  and  $R_{FFFS}(k, m)$  respectively, and  $R_{HSW}(k_1, k, m)$  for policy HSW, where  $k$  of the  $m$  machines are special and  $k_1$  of the  $k$  special windows are set to serve special customers only.

Using the worst-case analysis method, we get

$$\begin{aligned} R_{FCFS}(k, m) &= 1 + \frac{m-1}{k} \text{ for } 1 \leq k \leq m, \\ R_{SWSC}(k, m) &= 1 + \frac{m-1}{m-k} \text{ for } 1 \leq k < m, \\ R_{FFFS}(k, m) &= 1 + \frac{m-1}{k} \text{ for } \frac{m}{2} \leq k \leq m, \\ R_{FFFS}(k, m) &= 4 - \frac{2k}{m} - \frac{1}{k} \text{ for } 1 \leq k \leq \frac{m}{2}. \end{aligned}$$

We also have

(i) if  $2 \leq k \leq \frac{m}{2}$ ,

$$R_{HSW}(k_1, k, m) = \begin{cases} 2 + 2 \frac{(k-k_1)(m-k)}{k(m-k_1)} - \frac{1}{k} & \text{if } 0 \leq k_1 \leq \frac{m(k-1)}{2m-k-1} \\ 1 + \frac{k-1}{k_1} & \text{if } \frac{m(k-1)}{2m-k-1} \leq k_1 \leq \frac{m(k-1)}{m+k-2} \\ 1 + \frac{m-1}{m-k_1} & \text{if } \frac{m(k-1)}{m+k-2} \leq k_1 \leq k, \end{cases}$$

(ii) if  $\frac{m}{2} < k \leq \frac{2}{3}m$ ,

$$R_{HSW}(k_1, k, m) = \begin{cases} 1 + \frac{m-1}{k} & \text{if } 0 \leq k_1 \leq 2k-m \\ 2 + 2 \frac{(k-k_1)(m-k)}{k(m-k_1)} - \frac{1}{k} & \text{if } 2k-m \leq k_1 \leq \frac{(2k-1)(m-k)}{2m-k-1} \\ 1 + \frac{m-1}{m-k_1} & \text{if } \frac{(2k-1)(m-k)}{2m-k-1} \leq k_1 \leq k, \end{cases}$$

(iii) if  $\frac{2m}{3} < k < m$ ,

$$R_{HSW}(k_1, k, m) = \begin{cases} 1 + \frac{m-1}{k} & \text{if } 0 \leq k_1 \leq m-k, \\ 1 + \frac{m-1}{m-k_1} & \text{if } m-k \leq k_1 \leq k, \end{cases}$$

Comparing the tightly upper bounds of the four service policies, we get the following theorem.

**Theorem 1** (i) When  $k = 1$ , we have

$$R_{FCFS}(k, m) \geq R_{FFFS}(k, m) \geq R_{SWSC}(k, m).$$

(ii) When  $2 \leq k \leq \frac{m}{2}$ , we have

$$R_{FCFS}(k, m) \geq R_{FFFS}(k, m) \geq R_{SWSC}(k, m) > R_{HSW}(k^*, k, m),$$

where  $k^*$  is equal to either  $l_1 = \lfloor \frac{m(k-1)}{m+k-2} \rfloor$  or  $l_2 = \lceil \frac{m(k-1)}{m+k-2} \rceil$  such that

$$R_{HSW}(k^*, k, m) = \min\{R_{HSW}(l_1, k, m), R_{HSW}(l_2, k, m)\}.$$

(iii) When  $\frac{m}{2} < k \leq \frac{2m}{3}$ , we have

$$R_{SWSC}(k, m) \geq R_{FCFS}(k, m) = R_{FFFS}(k, m) \geq R_{HSW}(k^*, k, m),$$

where  $k^*$  is equal to either  $l_1 = \lfloor \frac{(2k-1)(m-k)}{2m-k-1} \rfloor$  or  $l_2 = \lceil \frac{(2k-1)(m-k)}{2m-k-1} \rceil$  such that

$$R_{HSW}(k^*, k, m) = \min\{R_{HSW}(l_1, k, m), R_{HSW}(l_2, k, m)\}.$$

$R_{HSW}(k^*, k, m)$  is strictly less than  $R_{FFFS}(k, m)$  if  $2k - m + 1 \leq l_1$ .

(iv) When  $\frac{2m}{3} < k < m$ , we have

$$R_{SWSC}(k, m) \geq R_{FCFS}(k, m) = R_{FFFS}(k, m) = R_{HSW}(k_1, k, m),$$

where  $0 \leq k_1 \leq m - k$ .

With Theorem 1, a system manager, without of any knowledge of operations research, can easily give the optimal strategy to set all the service windows if the queueing systems are in a rush period or in an emergency case. When  $k = 1$ , use ‘the special window serves special customers’ policy. When  $2 \leq k \leq \frac{m}{2}$ , first calculate  $l_1 = \lfloor \frac{m(k-1)}{m+k-2} \rfloor$  and  $l_2 = \lceil \frac{m(k-1)}{m+k-2} \rceil$ , then select  $k^* = \operatorname{argmin}\{1 + \frac{k-1}{l_1}, 1 + \frac{m-1}{m-l_2}\}$ , finally set  $k^*$  special windows using ‘special windows serve special customers’ policy, set the remaining  $m - k^*$  windows using ‘first fit, first serve’ policy. When  $\frac{m}{2} < k \leq \frac{2m}{3}$ , first calculate  $l_1 = \lfloor \frac{(2k-1)(m-k)}{2m-k-1} \rfloor$  and  $l_2 = \lceil \frac{(2k-1)(m-k)}{2m-k-1} \rceil$ , then select  $k^* = \operatorname{argmin}\{2 + 2\frac{(k-l_1)(m-k)}{k(m-l_1)} - \frac{1}{k}, 1 + \frac{m-1}{m-l_2}\}$ , and finally set  $k^*$  special windows using ‘special windows serve special customers’ policy, set the remaining  $m - k^*$  windows using ‘first fit, first serve’ policy. When  $\frac{2m}{3} < k < m$ , use ‘first come, first serve’ policy.

**Key Words:** service policy evaluation, parallel machine scheduling, the worst-case analysis.



# Integrated Scheduling of Different Types of Handling Equipment at Automated Container Terminals

Henry, Y. K. Lau<sup>a</sup>, Ying Zhao<sup>a</sup> and Chuanyou Pan<sup>b</sup>

<sup>a</sup>Department of Industrial and Manufacturing Systems Engineering  
The University of Hong Kong  
Pokfulam Road, Hong Kong, PRC  
hyklau@hku.hk, [zhao-ying@hkusua.hku.hk](mailto:zhao-ying@hkusua.hku.hk)  
<sup>b</sup>ZTE corporation, Shenzhen, PRC  
[pan.chuanyou@zte.com.cn](mailto:pan.chuanyou@zte.com.cn)

## Abstract

Effective control and operation of handling equipment in port container terminals are crucial in enhancing the overall efficiency and performance of container handling in import, export and transshipment. Nowadays, more and more container terminals are deploying automated handling equipment to improve the overall productivity. As the equipment is often unmanned and automatically controlled, it is essential to have a formal scheduling method for generating exact work orders, preferably in a dynamic nature, to instruct the equipment to perform the corresponding tasks in real-time. Moreover, since operator intervention is not available online, efficient planning and scheduling are crucial to achieve satisfactory overall performance.

In automated material handling, most existing literature deals with the scheduling of a single type of equipment, for instance the scheduling of automated guided vehicles (AGVs) and transfer cranes. However, due to the nature of material handling operations, the schedules of different types of equipment are highly interactive. In practice, the main loss of efficiency appears to be resulted from the schedules of automated equipment such as AGVs, stacker cranes (SCs), cargo hoists, conveyors, etc., which are not coordinated optimally. Most often, the individual or disjointed scheduling of the equipment may lead to low overall performance or even resulted in deadlocking situations. To address the very issue of optimal performance of a complete material handling task, this research deals with the integrated scheduling of more than one type of handling equipment with double cycle operations, in which the loading and the discharging operations can be performed alternatively. With this mode of operation, a multi objective optimization of the cooperating entities can be effectively performed in order to achieve an overall optimal performance of container handling in an automated port container terminal.

An automated container terminal with three types of equipment, quay cranes (QCs), automated guided vehicles (AGVs) and automated yard cranes (AYCs) is considered in this study. Our objective is to construct a detailed schedule for all the QCs, AGVs and AYC, such that the delay time of QCs, and the travel time of AGVs and AYC are minimized. As the interaction between QCs, AGVs and AYC has a great influence to the container handling process, the cooperating events are the basic scheduled objects. Therefore, the schedules we constructed determine the timing of the cooperating events between QCs and AGVs. Moreover, the cooperating events must be assigned to individual AGVs and the order and the timing of the events assigned to the same AGV must also be determined. Finally, for each of the AYC, as the corresponding tasks assigned are already known (since it is known a prior the specific stack lane a container is to be stored), our task is to determine the order and the timing of the cooperating events between the AGVs and the individual AYC.

A mixed-integer programming model is provided for the integrated scheduling problem with the objectives of minimizing the total delay time of QCs, and the total travel time of AGVs and AYC. This problem is a NP-complete problem, for which the polynomial time is unlikely. To facilitate the solution procedure, a heuristic algorithm called multi-layer genetic algorithm (MLGA) is proposed.

The main-layer genetic algorithm is to find the optimal sequence of the cooperating events between QCs and AGVs, while the sub-layer aims to find the sequence of the cooperating events of the individual AGVs, which actually deals with the AGV task assignment problems. As the sequence of the cooperating events between AGVs and individual AYC is a subset of the optimal sequence of the cooperating events between QCs and AGVs obtained, this integrated scheduling problem is solved by the proposed MLGA. In the MLGA that we developed, the chromosomes are represented as order-

based character strings, while in the sub-layer, the representation consists of a *event symbol* list and an *assignment symbol* list, in which the integers are used to represent all possible permutation of events, and asterisks \* are used to designate the assignment of events to AGVs. Two order based crossover; namely, partially mapped crossover (PMX) and order crossover (OX) are adopted for the main-layer GA. While in the sub-layer GA, the crossover takes two parents and creates a single child by propagating the overall partitioning structure and a sub-schedule into a child from one parent and then completing the child with the remaining tasks derived from another parent. This paper adopts the swapping mutation operation, which selects two random positions and then swaps their genes.

The MLGA provides near optimal solutions within a short computation time when the number of tasks is relatively small. However, the computational time increases rapidly when the problem size increases, which makes the MLGA impractical to be used in practice. Therefore, by revising the GA on the sub-layer, a heuristic algorithm called genetic algorithm plus maximum matching (GAPM) is suggested.

With the GAPM algorithm, the problem of assigning the sequence of the cooperating events to the individual AGVs is represented by a bipartite graph with the supply nodes including the set of event sequences generated by the GA and nodes that represent the starting events of the AGVs, the demand nodes include the set of event sequence generated by the GA and the stopping events of the AGVs. Assuming that each AGV can perform only one operation at a time; the assignment task is performed by assigning each node in the supply-side to a node in the demand-side in a one-to-one manner. If a node in the demand-side can not be matched to any nodes in the supply-side, this sub-problem is infeasible. In order to make an infeasible sub-problem feasible, one or more arcs have to be added to the bipartite graph. As one of the objectives of the scheduling problem is to minimize the delay of QCs' operations, the arc with a minimum additional delay is added to the graph. When a sub-problem becomes feasible, the node in the demand-side must be matched to the node in the supply-side with the minimum AGV travel time. Moreover, the total travel time of individual AYC is determined by the event sequence generated from GA.

In this study, three different experiments were performed. The first experiment compared two order based crossover operators. Although the two heuristic algorithms using the PMX outperformed that using the OX based on the computational times, their objective values were similar. The second experiment was set to search for the best set of parameters, for instance, the population size, and the mutation and crossover rate. The third experiment was performed by taking the better parameters obtained in the second experiment to compare the performances of the two heuristic algorithms. The results show that the objective values of GAPM are less than 5% of those of the MLGA, while the computational times required by GAPM is much reduced to 13.5% of those of MLGA on average. The result of much improved computation efficiency of GAPM suggests that GAPM is much suitable for real-time applications in the real world.

# **Supply Chain Scheduling**

Full Papers

# SINGLE MACHINE MULTIPLE ORDERS PER JOB SCHEDULING USING COLUMN GENERATION

Jagadish Jampani, Scott J. Mason, Vishnu Erramilli

*Department of Industrial Engineering, 4207 Bell Engineering Center, University of Arkansas, Fayetteville, AR 72701, USA*

**Abstract:** Effective production scheduling in 300-mm semiconductor wafer fabs is a challenging problem. Customers order integrated circuits, hundreds of which can be fabricated on a single silicon wafer 300-mm in diameter. Front open unified pods (FOUPs) are the basic units of wafer storage and transport in these newest semiconductor fabs. To achieve production efficiencies, the silicon wafers often must be grouped together in FOUPs without regard to the originating customer who placed the order. In the resulting *multiple orders per job (MOJ) scheduling problem*, scheduling is performed at the FOUP (i.e., aggregated order) level, but customer delivery performance is measured at the individual customer order level. Previous research efforts have illustrated the difficulty of obtaining optimal solutions for even small MOJ problems. We present a column generation-based heuristic approach for analyzing the single machine MOJ scheduling problem under two different objective functions: makespan and total weighted completion time. Experimental results demonstrate the proposed column generation approach's ability 1) to obtain good solutions to MOJ problems in a fairly short amount of computation time (on average) and 2) to outperform competing approaches in the literature for 50-order single machine MOJ scheduling problem instances.

**Key words:** semiconductor manufacturing, column generation, heuristic

## 1. INTRODUCTION

The last decade has seen tremendous growth in the semiconductor manufacturing industry. With the ever increasing demand for semiconductor based products, the industry is trying to meet ever tightening customer deadlines while simultaneously trying to evolve into more efficient production units. There is a healthy competition in the market with every manufacturer trying to increase their market share by meeting deadlines and improved services to the customers. The demand of on-time delivery of high quality products has motivated the manufacturers to implement technologies based on dispatching and scheduling to reduce the production time. Efficient scheduling of the available resources is an effective method to achieve the organizational goals of on-time deliveries of products at a low production cost.

Semiconductor manufacturing can be divided into four phases: wafer fabrication, wafer probe, assembly and final testing (Uzsoy *et al.* 1992). The wafer fabrication phase is the most complex phase with a number of process flows, each with 300-500 steps associated with it. The equipment used in wafer fabs is sophisticated and complex. Unique operating characteristics of the wafer fabs like re-entrant flows, batch processing, sequence dependent set-ups, etc., all add to the complexity of the wafer fabs. Due to the above factors Mason *et al.* (2002) refer to wafer fabs as complex job shops.

A part of the evolution in semiconductor manufacturing industry is the advancement in the processes and machine tools required to produce the products. New technologies are being introduced in the wafer fabrication equipment and production processes. One of the main changes has been the adoption of 300-mm diameter wafer in place of the earlier standard of 200-mm. Due to increase in diameter of the wafers more die can be obtained per wafer. This result in increment in the number of integrated circuits produced per wafer, and better utilization of the highly capital-intensive machinery. But the increase in size has also brought about an increase in weight of the basic wafer. The 300-mm wafer cannot be manually moved between work stations therefore complete automation of the production system is required. A part of this automation is the usage of entities called front-open unified pods (FOUPs). FOUPs provide an inert, nitrogen atmosphere thus preventing the contamination of surfaces of the wafers.

Customer orders are translated in terms of the number of wafers required to fulfil the order. Each order can be assigned its own FOUP while production but the idea is not financially viable due to the high cost associated with these FOUPs. Also, if each order would be assigned its own FOUP, then there would be too many FOUPs in the system causing the automated material handling system (AMHS) to be overloaded. To overcome these challenges one needs to group these multiple orders into the FOUPs (can also be considered as a job). Once the orders have been grouped into jobs, these jobs can be scheduled to ensure efficient usage of the resources available in the fab.

The research in this paper has been mainly motivated by this challenge of grouping different customer orders each with different attributes into single jobs and then scheduling these formed jobs on a single machine. The objectives of the scheduling problem investigated in this research are minimizing the makespan (i.e., production cycle time) and total weighted order completion time (i.e., throughput maximization when differing customer priorities exist). The single lot MOJ processing environment is studied in which the processing time of the job is independent of its contents. All the wafers in the job are simultaneously processed and therefore all the orders have the same processing time on the considered single machine, such as is the case in a wet sink or acid bath. The  $1|moj(lot)|C_{\max}$  and  $1|moj(lot)|\sum w_j C_j$  scheduling problems are NP hard, as the known NP hard bin packing problem

(Karp 1972) is reducible to both of these problems, as clearly in both MOJ problem cases, we are seeking to minimize the number of FOUPs (i.e., bins) that the customer orders are assigned to.

The following sections of this paper are organized as follows. Section 2 reviews the relevant literature to our problem. Next, the mathematical formulation and the solution methodology are presented and explained in Section 3, followed by a comparison of the solutions obtained from our proposed solution methodology and the known optimal solutions in Section 4. Finally, we make concluding remarks and offer directions for future research in Section 5.

## 2. LITERATURE REVIEW

Scheduling jobs containing multiple orders is a fairly new field of research and is practically motivated by the recent developments in the semiconductor manufacturing industry. Kutanoglu *et al.* (2004) address this problem of job formation and scheduling considering a special case of the main problem with an objective of minimizing the total completion time of the orders. It is assumed that there is no restriction on the number of FOUPs available in the system. An algorithm is introduced and preliminary computational results are reported. Results were presented for problems containing up to 24 orders and six jobs/FOUPs.

Mason *et al.* (2004) propose an optimization model to solve this *multiple orders per job* (MOJ) scheduling problem with an objective of minimizing the total weighted completion time of the orders. Both single lot and single item processing cases are considered. In the single item (single lot) processing case, job processing time is dependent on (independent of) the wafers present in the FOUP. A non-linear mathematical model is developed which is linearized by assuming that the sequence of the jobs is

predefined. They are sequenced according to the increasing order of their indices. It is shown that for the cases above 15 orders, the optimization model takes impractical amounts of computation time. Hence, some heuristic methods, based on the first fit decreasing approach for solving the traditional bin-packing problems, are presented to solve larger problem instances. It is observed that the heuristics perform 1% - 2% above the optimal solution on average for the 10- and 15-orders cases.

Single machine scheduling problems and their solution approaches are addressed in Pinedo (1995). Van den Akker *et al.* (2000) presents a formulation of time-indexed single machine scheduling problems and use the column generation method to solve it. In this formulation, assigning the jobs to the machines is considered with an objective of minimizing the total cost. In the literature, column generation method has been effectively used to solve large scale problems like cutting stock, aircraft routing, ship scheduling, vehicle routing, bin packing, network flow, and circuit card assembly.

### 3. MATHEMATICAL FORMULATION AND SOLUTION METHODOLOGY

We assume that at an instance of time, we have a set of orders ( $O$ ) that are placed by the customers. Depending on the customers demand and importance, an order ( $o \in O$ ) has a size  $s_o$  and weight  $w_o$  respectively. It is assumed that all the sizes of the orders are less than or equal to the maximum FOUP capacity. All orders are to be placed in the FOUPs and scheduled to attain the required objective. As a lot processing situation is considered, it is assumed that each FOUP takes ten units of processing time (ten units of processing time is an arbitrary value and can be changed depending on the situation). As ready times of the orders are not considered, it is assumed that any of the orders can be selected to be placed in the FOUP at any chosen instance of time.

#### 3.1 Solution Methodology

In the MOJ scheduling problem, we have a large number of variables and integrality constraints on the variables, both of which combine to result in impractical computation time. Column generation (*CG*) method (Dantzig *et al.* (1960)) is proposed to solve this problem. The problem is divided into two interrelated models; the master problem (*MP*) and sub problem (*SP*). The *MP* is relaxed and restricted because non-integer values can be taken by the decision variables in *MP* and only the columns that have the ability to improve the objective of the *MP* are generated in the *SP*. During each iteration of the *CG* procedure, the *MP* is solved to optimality and by using its dual information, the *SP* generates a new column. This new column contains the information regarding which order goes into which job or FOUP. This iterative process stops when no improving columns are generated by the *SP*.

#### 3.2 A Heuristic for $1 | moj(lot) | C_{\max}$

The following discrete optimization model minimizes the makespan of customer orders. Each column in the basis matrix represents a pattern, which indicates a way of placing different orders in a FOUP. First, we define pertinent notation:

- $o$  Index over the total number of orders  $O$  ( $o \in O$ )
- $j$  Index over the total number of patterns or columns  $J$  ( $j \in J$ )
- $K$  Maximum FOUP capacity
- $\rho_o$  Processing time of a job; assumed to be equal to 10 time units for all jobs and hence orders.
- $A_{o,j}$  Number of times an order  $o$  appears in pattern  $j$

$D_o$  Demand of each order  $o$

$X_j$  Number of times the pattern  $j$  is selected (variable in *MP1*)

$Y_o = 1$ , if order  $o$  appears in a particular pattern, 0 otherwise (variable in *SP1*).

Now, we describe our  $1 | \text{moj}(\text{lot}) | C_{\max}$  master problem:

*Restricted Master Problem (MP1):*

$$\text{Minimize} \quad \rho_o \sum_{j \in J} X_j \quad (1)$$

$$\text{s.t.} \quad \sum_{j \in J} A_{o,j} X_j \geq D_o \quad \forall o \in O \quad (2)$$

$$X_j \geq 0 \quad \forall j \in J \quad (3)$$

The objective (1) in *MP1* selects the minimum number of patterns satisfying the demand constraints (2) for all the orders. The objective is also equivalent to selecting the minimum number of FOUPs required to fill all customer orders. The sub problem (*SP1*) is solved using the dual variable information from *MP1*. Parameters  $\pi_o$  represent the dual values corresponding to constraint (2), which are non-negative in sign. New columns for *MP1* are generated by solving the simple knapsack problem as the sub problem.

*Sub Problem (SP1):*

$$\text{Maximize} \quad \sum_{o \in O} \pi_o Y_o \quad (4)$$

$$\text{s.t.} \quad \sum_{o \in O} s_o Y_o \leq K \quad (5)$$

$$Y_o \in \{0,1\} \quad \forall o \in O \quad (6)$$

### 3.3 A Heuristic for $1 | \text{moj}(\text{lot}) | \sum_{o \in O} w_o C_o$

This subsection contains a discrete optimization model for minimizing total weighted completion time of customer orders. In this formulation, an upper bound on the number of FOUPs  $T$  is calculated, which is equal to the number of FOUPs in the input feasible solution for the column generation method. In a feasible solution, all orders are selected once and placed in FOUPs while satisfying the FOUP capacity constraint. A FOUP is divided into  $K$  slots, where  $K$  equals the maximum FOUP capacity. Further, only one wafer is allowed to occupy a slot. This avoids an extra constraint in the formulation regarding the FOUP capacity. In this formulation, a generated column has information that represents the slots occupied by wafers in each FOUP. First, we define pertinent notation:

$o$  Index over the total number of orders  $O$  ( $o \in O$ )

$k$  Index over the slot number in a FOUP ( $k \in K$ )

$t$  Index over the upper bound on the number of FOUPs ( $t \in T$ )

$c$  Index over the columns or patterns

$C(o)$  Columns corresponding to order  $o$

$W_{o,c}$  Product of order  $o$ 's weight ( $w_o$ ) and the FOUP number ( $t$ ) it is placed in the column  $c$

$A_{o,k,t,c} = 1$ , if order  $o$  occupies slot  $k$  in FOUP  $t$  in column  $c$ , 0 otherwise

$X_{o,c} = 1$ , if order  $o$  is in column  $c$ , 0 otherwise (Variable in *MP2*)

$Y_{o,k,t,c} = 1$ , if order  $o$  occupies slot  $k$  in FOUP  $t$  in column  $c$ , 0 otherwise (Variable in *SP2*)

$G_t = 1$ , if an order is present in foup  $t$ , 0 otherwise (Variable in *SP2*)

Now, we describe our  $1 | \text{moj}(\text{lot}) | \sum_{o \in O} w_o C_o$  master problem:

*Restricted Master Problem (MP2):*

$$\text{Minimize} \quad \rho_o \sum_{o \in O} \sum_{c \in C(o)} W_{o,c} X_{o,c} \quad (7)$$

$$\text{s.t.} \quad \sum_{c \in C(o)} X_{o,c} = 1 \quad \forall o \in O \quad (8)$$

$$\sum_{o \in O} \sum_{c \in C(o)} A_{o,k,t,c} X_{o,c} \leq 1 \quad \forall t \in T, \forall k \in K \quad (9)$$

$$X_{o,c} \geq 0 \quad \forall o \in O, \forall c \in C \quad (10)$$

The objective function (7) minimizes the weighted completion times of the orders. Constraint (8) makes sure that every order is selected once and constraint (9) limits a slot in all the FOUPs to be occupied by only one wafer or remains empty.

Solving *MP2*, dual variables corresponding to the constraints are obtained. The set of dual values  $\nu_o$  are unrestricted in sign and represent the dual values of the set of constraints (8). The set of dual values  $\pi_{t,k}$  are negative in sign and correspond to the set of constraints (9). These dual variables are used in objective function of the sub problem to assign the costs of the slots in the FOUPs. The sub problem is solved for a number of times equal to the number of orders, generating a new column for each order. All these newly generated columns are added to the *MP2*. This sub problem can be seen as a simple assignment problem which minimizes the cost of selecting/assigning slots in the FOUPs to the orders.

*Sub-Problem (SP2):*

$$\text{Minimize} \quad \sum_{t \in T} \sum_{k \in K} (((w_o / s_o)t) - \pi_{t,k}) Y_{o,k,t,c} - \nu_o \quad (11)$$

$$\text{s.t.} \quad \sum_{t \in T} \sum_{k \in K} Y_{o,k,t,c} = s_o \quad (12)$$

$$\sum_{k \in K} Y_{o,k,t,c} - s_o G_t = 0 \quad \forall t \in T \quad (13)$$

$$Y_{o,k,t,c} \in \{0,1\} \quad \forall o \in O, \forall k \in K, t \in T, c \in C \quad (14)$$

The *SP2* objective function (11) minimizes the cost of placing the orders in the FOUPs. This cost is equal to the product of the weight of an order  $o$  ( $w_o$ ) and the FOUP number ( $t$ ) the order is placed in. Constraint (12) constrains that slots equal to the size of an order are occupied over all the FOUPs. As orders are not allowed to be split across the FOUPs, constraint (13) is introduced so that either a complete order is placed in a FOUP or nothing related to that order is placed in it.



#### 4. EXPERIMENTAL RESULTS

In order to evaluate our column generation-based heuristics, an experimental design was developed (Table 1). The order sizes are generated using two discrete uniform functions;  $s_o \sim \text{DU}[1,5]$  when  $\nu = 3$  and  $s_o \sim \text{DU}[2,8]$  when  $\nu = 5$ . The FOUP capacity is varied over two levels;  $K = 13$  and  $K = 25$ . The weights of the orders are generated using the discrete uniform function  $w_o \sim \text{DU}[1,15]$ . For each combination of the order size and FOUP capacity, 10 replications are solved. Hence, varying the two factors for a 10 order problem instance, 40 computation results are obtained. This is repeated for the 15, 20 and 50 order problems.

The computations are carried on a computer with 3.4 GHz processor and 2 GB of RAM. The solutions obtained from the optimization model of Mason *et al.* (2004) and the proposed column generation methods are compared for the 10, 15 and 20-order problems. As the optimization model is not able to obtain solutions for the 50 order problems in reasonable amount of time, the best of the heuristic solutions from Mason *et al.* (2004) is used for comparison. The performance ratio of the column generation method is calculated by dividing the column generation solution by the optimal solution or best heuristic in the 50-order case. The best, the worst and the average performance ratios for the combination of each of the levels of the factors are shown in Tables 2 and 3 for makespan and Tables 4 and 5 for total weighted completion time.

With an objective of minimizing the makespan (Tables 2 and 3), it is observed that the proposed column generation-based heuristic performs comparably to the heuristics of Mason *et al.* (2004). In terms of minimizing weighted completion time, the column generation's average performance ratio varies between one and 2%, with an average value of 1.6%. This performance is slightly better than the best previous heuristic solution of Mason *et al.* (2004). As was the case in the makespan instances, the column generation approach finds many of the known optimal solution values. For the 50-order total weighted completion time cases, the column generation approach again outperforms Mason *et al.* (2004). However, this improved performance can come at a computational cost.

The average computation times for all the problem instances are presented in Tables 6 and 7. It can be observed that the column generation method obtained good solutions in reasonable amount of computation time. Unfortunately, the average computation time for the 50-order problem instances is relatively high; however, it varies widely depending on the problem instance. The computation time also depends on how "good" the initial columns are. As the FOUP size increases from 13 to 25, the length of the generated column increases resulting in a considerable increase in computation time.

#### 5. CONCLUSIONS AND FUTURE RESEARCH

In this paper, the MOJ scheduling problems of 1) minimizing makespan and 2) minimizing total weighted order completion time are addressed. Mathematical models are developed and solved using a column generation heuristic approach. Experimentation conducted over a wide range of problem instances demonstrated the efficacy of the proposed column generation approach. The new method, although outperforming the best heuristics to date for large 50-order problem instances, can result in an unacceptable amount of computation time due to increased subproblem basis matrix dimensionality.

This research can be extended to address the single item processing environment in MOJ scheduling in which job processing time depends on the number of items in the job. Future research efforts will pursue this extension, as well as the inclusion of order ready times and job setup times and more advanced machine and shop environments in order to increase the viability of our heuristic approaches for deployment in real world 300-mm wafer fabs.

## REFERENCES

- Abara J. (1989), Applying integer linear programming to the fleet assignment problem, *Interfaces*, 19, 4, 20-28.
- Cattrysse, D.G., Salomon, M., Van Wassenhove, L.N., (1994), A set partitioning heuristic for the generalized assignment problem, *European Journal of Operational Research*, 72, 1, 167-174.
- Dantzig, G.B., and Wolfe, P., (1960) Decomposition principle for linear programs, *Operations Research*, 8, 1, 101-111.
- Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F., (1996), Time constrained vehicle routing and scheduling, *Handbooks in Operations Research and Management Science*, Vol. 8: Network Routing Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L., eds., North-Holland: Amsterdam, 35-140.
- Gilmore, P.C., Gomory, R.E., (1963), A linear programming approach to the cutting stock problem-Part II, *Operations Research*, 11, 6, 863-888.
- Karp, R.M., (1972), Reducibility among combinatorial problems, In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York, 85-103.
- Kutanoglu, E., Tanrisever, F., Mason, S.J. (2004) Forming and scheduling jobs from multiple orders with different attributes: a computational study of special cases, *13th Annual Industrial Engineering Research Conference*, Houston, Texas.
- Mason, S.J., Fowler, J.W., Carlyle, W.M. (2002) A modified shifting bottleneck heuristic for minimizing the total weighted tardiness, *Journal of Scheduling*, 5, 3, 247-262.
- Mason, S.J., Qu, P., Kutanoglu, E., Fowler, J.W. (2004) The single machine multiple orders per job scheduling problem, *IIE Transactions*, in review.
- Pinedo. M. (1995) *Scheduling: Theory, Algorithms and Systems*, John Wiley: Englewood Cliffs, NJ 1995.
- Uzsoy, R., Lee, C.Y., Martin-Vega, L.A. (1992) A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part I: System Characteristics, Performance Evaluation and Production Planning, *IIE Transactions*, 24, 47-60.
- Van den Akker, J.M., Hurkens, C.A.J., Salvesbergh, M.W.P. (2000) A time-indexed formulation for single-machine scheduling problems: Column generation, *INFORMS Journal on Computing*, 12, 2, 111-124.
- Wilhelm, E.W. (2001) A technical review of column generation in integer programming, *Optimization and Engineering*, 2, 2, 159-200.

Table 1. Experimental design space for the lot processing instances

| Factors       | Levels | Levels Description                                                  |
|---------------|--------|---------------------------------------------------------------------|
| Order Size(Z) | 2      | Discrete Uniform                                                    |
|               |        | $DU[\nu - (\nu + 1)/2, \nu + (\nu + 1)/2]$ where $\nu \in \{3, 5\}$ |
| FOUP          | 2      | $12\beta + 1$ where $\beta \in \{1, 2\}$                            |
| Capacity(K)   |        |                                                                     |

Table 2. Performance ratios of 10 and 15-orders  $1 | \text{moj}(\text{lot}) | C_{\max}$  problem instances

|                |           | 10 Orders |       |              | 15 Orders |       |              |
|----------------|-----------|-----------|-------|--------------|-----------|-------|--------------|
|                |           | Best      | Worst | Average      | Best      | Worst | Average      |
| $K = 13$       | $\nu = 3$ | 1.000     | 1.500 | 1.050        | 1.000     | 1.000 | 1.000        |
|                | $\nu = 5$ | 1.000     | 1.000 | 1.000        | 1.000     | 1.000 | 1.000        |
| $K = 25$       | $\nu = 3$ | 1.000     | 1.000 | 1.000        | 1.000     | 1.166 | 1.016        |
|                | $\nu = 5$ | 1.000     | 1.000 | 1.000        | 1.000     | 1.000 | 1.000        |
| <b>Average</b> |           |           |       | <b>1.012</b> |           |       | <b>1.004</b> |

Table 3. Performance ratios of 20 and 50-orders  $1 | \text{moj}(\text{lot}) | C_{\max}$  problem instances

|                |           | 20 Orders |       |              | 50 Orders |       |              |
|----------------|-----------|-----------|-------|--------------|-----------|-------|--------------|
|                |           | Best      | Worst | Average      | Best      | Worst | Average      |
| $K = 13$       | $\nu = 3$ | 1.000     | 1.200 | 1.056        | 1.000     | 1.090 | 1.009        |
|                | $\nu = 5$ | 1.000     | 1.222 | 1.055        | 1.000     | 1.000 | 1.000        |
| $K = 25$       | $\nu = 3$ | 1.000     | 1.000 | 1.000        | 1.000     | 1.555 | 1.020        |
|                | $\nu = 5$ | 1.000     | 1.250 | 1.115        | 1.000     | 1.111 | 1.011        |
| <b>Average</b> |           |           |       | <b>1.056</b> |           |       | <b>1.010</b> |

Table 4. Performance ratios of 10 and 15-orders  $1 | \text{moj}(\text{lot}) | \sum w_j C_j$  problem instances

|                |           | 10 Orders |       |              | 15 Orders |       |              |
|----------------|-----------|-----------|-------|--------------|-----------|-------|--------------|
|                |           | Best      | Worst | Average      | Best      | Worst | Average      |
| $K = 13$       | $\nu = 3$ | 1.000     | 1.009 | 1.001        | 1.000     | 1.020 | 1.009        |
|                | $\nu = 5$ | 1.000     | 1.070 | 1.025        | 1.007     | 1.053 | 1.017        |
| $K = 25$       | $\nu = 3$ | 1.000     | 1.000 | 1.000        | 1.000     | 1.012 | 1.001        |
|                | $\nu = 5$ | 1.000     | 1.070 | 1.013        | 1.000     | 1.090 | 1.024        |
| <b>Average</b> |           |           |       | <b>1.010</b> |           |       | <b>1.013</b> |

Table 5. Performance ratios of 20 and 50-orders  $1 | \text{moj}(\text{lot}) | \sum w_j C_j$  problem instances

|                |         | 20 Orders |       |              | 50 Orders |       |              |
|----------------|---------|-----------|-------|--------------|-----------|-------|--------------|
|                |         | Best      | Worst | Average      | Best      | Worst | Average      |
| $K = 13$       | $v = 3$ | 1.000     | 1.017 | 1.006        | 0.952     | 1.006 | 0.994        |
|                | $v = 5$ | 1.000     | 1.028 | 1.008        | 0.611     | 0.994 | 0.950        |
| $K = 25$       | $v = 3$ | 1.000     | 1.014 | 1.004        | 0.994     | 1.002 | 0.998        |
|                | $v = 5$ | 1.000     | 1.053 | 1.014        | 0.994     | 1.000 | 0.997        |
| <b>Average</b> |         |           |       | <b>1.008</b> |           |       | <b>0.984</b> |

Table 6. Computation times of  $1 | \text{moj}(\text{lot}) | C_{\max}$  problem instances in seconds

|         |          | 10     | 15     | 20     | 50     |
|---------|----------|--------|--------|--------|--------|
|         |          | Orders | Orders | Orders | Orders |
| $v = 3$ | $K = 13$ | 0.159  | 0.207  | 4.141  | 0.306  |
|         | $K = 25$ | 0.153  | 0.329  | 1.979  | 0.306  |
| $v = 5$ | $K = 13$ | 0.201  | 0.120  | 11.467 | 0.635  |
|         | $K = 25$ | 0.254  | 0.123  | 8.591  | 0.298  |

Table 7. Computation times of  $1 | \text{moj}(\text{lot}) | \sum w_j C_j$  problem instances in seconds

|         |          | 10     | 15     | 20     | 50       |
|---------|----------|--------|--------|--------|----------|
|         |          | Orders | Orders | Orders | Orders   |
| $v = 3$ | $v = 13$ | 0.784  | 1.829  | 2.571  | 598.383  |
|         | $v = 25$ | 0.945  | 2.001  | 2.262  | 1387.038 |
| $v = 5$ | $v = 13$ | 1.737  | 4.820  | 9.131  | 1231.378 |
|         | $v = 25$ | 1.698  | 4.696  | 7.012  | 5117.444 |

# **Supply Chain Scheduling**

*Abstracts*

# Integrated Production-Distribution Scheduling to Optimize the Tradeoff between Order Delivery Performance and Total Cost

Guruprasad Pundoor  
FedEx Express  
3680 Hacks Cross Road  
Memphis, TN 38125

Zhi-Long Chen  
Department of Decision and Information Technologies  
Robert H. Smith School of Business, University of Maryland  
College Park, MD 20742-1815

The scheduling problem we consider is often faced by manufacturers who make time-sensitive products such as fashion apparel and toys, which typically have large product types and sell only during specific seasons. Consider the production and distribution scheduling decisions such a manufacturer needs to make. The customers (e.g. distributors or retailers) often set due dates on the orders they place with the manufacturer and there is typically a penalty imposed on the manufacturer if the orders are not completed and delivered to the customers on time. Hence the manufacturer would like to meet the due dates as much as possible. Another factor the manufacturer has to consider is the total cost for order processing and delivery. Since the products are time-sensitive, orders are delivered shortly after their completion and thus we assume that little inventory cost is incurred. The manufacturer's total cost is mainly contributed by production and distribution operations. The total production cost for a fixed set of orders is normally fixed and independent of the production schedule used. Therefore, the manufacturer should focus on the distribution cost when considering the total cost. Since different orders may have different due dates, delivering more orders on time might require the manufacturer to make a larger number of shipments leading to higher total distribution cost. Therefore, the manufacturer has to find a production and distribution schedule that achieves some balance between delivery timeliness and total distribution cost. In practice, the *maximum tardiness* of orders and the *total tardiness* of orders are the two commonly used measurements for delivery timeliness. They represent the worst and average service level with respect to meeting order due dates, respectively. In this paper, we focus on the maximum tardiness as the measurement for delivery timeliness.

Our model can be described more precisely as follows. There are one supplier and  $m$  ( $m \geq 1$ ) customers  $M = \{1, \dots, m\}$  located at different locations in a given production-distribution system. At the beginning of a planning horizon, the supplier receives  $n_i$  orders from each customer  $i \in M$ , requesting for processing. Let  $n = n_1 + \dots + n_m$  be the total number of orders. Let  $(i, j)$  denote the  $j$ th order from customer  $i$ ,  $N_i = \{(i, 1), \dots, (i, n_i)\}$  be the set of the orders from customer  $i$ , and  $N = N_1 \cup \dots \cup N_m$  be the set of all the orders. All the orders are to be processed on a single production line at the supplier. Each order  $(i, j) \in N$  has a processing time  $p_{ij}$  and a due date  $d_{ij}$ . Completed orders are delivered in batches to the customers. Due to the time sensitivity of the orders and the fact that each customer is located at a distinct location, direct shipping from the supplier to each customer is used. Therefore, only orders from the same customer can be batched together to form a delivery shipment and orders from different customers must be delivered separately. The delivery time and delivery cost from the supplier to customer  $i \in M$  are  $t_i$  and  $f_i$ , respectively. The maximum allowed batch size (i.e. the maximum number of orders that can be shipped in a batch) is given by  $b$ . Let  $C_{ij}$  and  $D_{ij}$  denote the processing completion time and delivery time of order  $(i, j) \in N$ , respectively. We define  $T_{ij} = \max\{0, D_{ij} - d_{ij}\}$  to be the tardiness of a particular order  $(i, j) \in N$  and  $T_{\max} = \max\{T_{ij} \mid (i, j) \in N\}$  to be the maximum tardiness of all orders. The total distribution cost for a given

schedule is denoted as  $G$ , and  $G = f_1x_1 + \dots + f_mx_m$ , where  $x_i$  is the number of shipments used to deliver the orders of customer  $i \in M$ .

The objective function should consider both  $T_{\max}$  and  $G$ . In order to achieve this, we define a weighting factor  $\alpha$  ( $0 \leq \alpha \leq 1$ ), which is based on the manufacturer's relative preference on the two measurements  $T_{\max}$  and  $G$ . The objective function is then defined as  $\alpha T_{\max} + (1-\alpha)G$ . It can be seen that when  $\alpha$  is close to 0, more emphasis is given to the total distribution cost. On the other hand, when  $\alpha$  is close to 1, more emphasis is given to  $T_{\max}$ . In situations where the relative preference on the two measurements  $T_{\max}$  and  $G$  is difficult to quantify, we can simply solve the problem multiple times with varying values of  $\alpha$  and pick one of the resulting solutions with the right level of balance between the two measurements  $T_{\max}$  and  $G$ .

In this talk, we will talk about the computational complexity of the problem under various cases. We give efficient algorithms for the problem under several special cases and show that the problem under the general case is NP-hard and give efficient algorithms for other cases of the problem. We will also talk about a quick heuristic for solving the general case of the problem. We show that the heuristic is asymptotically optimal with respect to the number of orders. To evaluate the performance of the heuristic, we develop a column generation based approach for generating lower bounds. Our computational experiment shows that the heuristic is capable of generating near optimal solutions. Finally, we will compare the performance of the integrated production-distribution approach with two sequential approaches that treat order processing and order delivery independent of each other. Managerial insights will be presented.

# Combined Production and Distribution Scheduling of Ready-Mix Concrete

**Pang Yoke Fong and Rajesh Piplani<sup>φ</sup>**  
Center for Supply Chain Management  
School of Mechanical and Aerospace Engineering  
Nanyang Technological University  
50 Nanyang Avenue, Singapore 639798

## ABSTRACT

The need for combined production and distribution scheduling is paramount in Ready Mixed Concrete (RMC) production as having an inventory of the finished product to meet the fluctuating demand is just not possible. This research aims to develop a feasible operating schedule for RMC production by addressing (simultaneously) the mixing of raw materials at the plants and the delivery of final product (concrete) to customer sites.

As the concrete delivery capacity, determined by truck availability, basically defines the capacity to fulfill customer orders, focus in the research is placed on vehicle scheduling, and the production operations are included as part of the delivery activities. A model for scheduling vehicles in a multi-depot scenario with added time constraints is developed and solved by Branch-and-Cut Algorithm. Several algorithm-related techniques are evaluated and subsequently enhanced to further improve the solution quality. The adequacy of the solution methodology to solve the model is further verified by comparing with heuristics-based search techniques within a Constraint Programming framework. With real-life data drawn from operational profiles of local RMC companies, five sets of practical-size instances of the model are developed and solved.

Application of Branch-and-Cut, with Strong Branching and Best Estimate rules, is able to find good integer feasible solutions, even for the largest instances, given enough time. In terms of computational efficiency, however, the exact method does not perform as well as the heuristic search techniques, where a local search procedure is found to be the most efficient. We also explore the feasibility of the generated schedule and potential advantages of the modeling approach in this research.

Future extensions to this work would include model and algorithmic developments to address issues such as larger problem instances, order allocation to plants and emergencies (sudden orders, order changes, etc.) in real-time. Finally, the developed model can be applied, with suitable modifications, to industries with similar characteristics of highly perishable products and multiple Just-In-Time deliveries.

**Keywords:** supply-chain scheduling, delivery scheduling, heuristic search.

---

<sup>φ</sup> Corresponding author, Ph. +65 6790 5601, Fax: +65 6791 1859, E-mail: mrpiplani@ntu.edu.sg



# INCREMENTAL PROPAGATION RULES FOR A PRECEDENCE GRAPH WITH OPTIONAL ACTIVITIES AND TIME WINDOWS

Roman Barták\*, Ondřej Čepek\*\*†

\*Charles University, Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic  
{roman.bartak, ondrej.cepek}@mff.cuni.cz

†Institute of Finance and Administration, Estonská 500, 101 00 Praha 10, Czech Republic

**Abstract:** Constraint-based scheduling is a powerful tool for solving real-life scheduling problems thanks to a natural integration of special solving algorithms encoded in global constraints. The filtering algorithms behind these constraints are based on propagation rules modelling some aspects of the problems, for example a unary resource. This paper describes new incremental propagation rules integrating a propagation of precedence relations and time windows for activities allocated to a unary resource. Moreover, the rules also cover so called optional activities that may or may not be present in the final schedule.

**Key words:** Constraint Logic Programming, Real World Scheduling

## 1. INTRODUCTION

Real-life scheduling problems usually include a variety of constraints so special scheduling algorithms (Brucker, 2001) describing a single aspect of the problem can hardly be applied to solve the problem completely. Constraint-based scheduling (Baptiste, Le Pape, Nuijten, 2001) provides a natural framework for modelling and solving real-life problems because it allows integration of different constraints. The above mentioned special scheduling algorithms can be often transformed into propagators for the constraints so the big effort put in developing these algorithms is capitalised in constraint-based scheduling.

Many filtering algorithms for specialised scheduling constraints have been developed in recent years (Baptiste, Le Pape, Nuijten, 2001). There exist algorithms based for example on edge-finding (Baptiste & Le Pape, 1996) or not-first/not-last (Torres & Lopez, 1997) techniques that restrict the time windows of the activities. Other algorithms are based on relative ordering of activities, for example filtering based on optimistic and pessimistic resource profiles (Cesta & Stella, 1997). Recently, as scheduling and planning technologies are coming together, filtering algorithms combining filtering based on relative ordering and time windows appeared. Detectable precedences by Vilím (2002) are one of the first attempts for such a combination. Laborie (2003) presents a similar rule called energy precedence constraint for reservoir-like resources.

Filtering algorithms for scheduling constraints typically assume that all the constrained activities will be included in the final schedule. This is not always true, for example assume that there are alternative processes to accomplish a job or alternative resources per activity. These alternatives are typically modelled using optional activities that may or may not be included in the final schedule depending on which process or resource is selected. The optional activity may still participate in the constraints but it should not influence other activities until it is known to be in the schedule. This could be realised by allowing the duration of the optional activity to be zero for time-windows based filtering like edge-finding (Baptiste, Le Pape, Nuijten, 2001). However, this makes filtering weaker and as shown in

(Vilím, Barták, Čepék, 2004) a stronger and faster filtering can be achieved if optional activities are assumed in the filtering algorithm directly. The paper (Focacci, Laborie, Nuijten, 2000) proposed a global precedence graph where alternative resources correspond to paths in the graph, but the graph is used merely for cost-based filtering (optimization of makespan or setup times).

In this paper we address the problem of integrated filtering based on precedence relations and time windows. From the beginning we assume the existence of optional activities. A filtering algorithm for these so called detectable precedences with optional activities on a unary resource has been proposed in (Vilím, Barták, Čepék, 2004). This algorithm uses  $\Theta$ - $\Lambda$ -tree to achieve  $O(n \log n)$  time complexity and it is a monolithic algorithm (must be repeated completely if there is any change of domains). The same pruning can be achieved by the energy precedence constraint proposed by Laborie (2003) if it is applied to a unary resource (the energy precedence constraint is defined for reservoirs). However, the energy precedence constraint is not defined for optional activities and details of implementation are not given in the paper.

We propose a new set of propagation rules that keep a transitive closure of the precedence relations, deduce new precedence relations, and shrink the time windows of the activities. They may also deduce that some optional activity will not be present in the final schedule. There are two main differences from the algorithm proposed in (Vilím, Barták, Čepék, 2004). First, we use “light” data structures, namely domains of variables. Second, the new rules are incremental so they directly react to changes of particular domains rather than running a monolithic algorithm from scratch. Such rules are much easier for implementation and for integration to existing constraint solvers and the hope is that their incremental nature will lead to a good practical efficiency. The implementation of the rules is currently being done so the paper reports a work in progress.

The paper is organised as follows. We first give more details on the problem to be solved. Then we describe the constraint services available for implementation of new constraints. In the main part of the paper, we describe a constraint-based representation of the precedence graph and we propose a set of propagation rules for the precedence graph. After that, we describe propagation rules for shrinking time windows by using information about precedence relations.

## 2. THE PROBLEM

In this paper we address the problem of modelling a unary resource where activities must be allocated in such a way that they do not overlap in time. We assume that there are time windows restricting the position of these activities. The time window  $[R, D]$  for an activity specifies that the activity cannot start before  $R$  (release time) and cannot finish after  $D$  (deadline). We assume the activity to be non-interruptible so the activity occupies the resource from its start till its completion, that is, for a time interval whose length is equal to the given length of the activity. We also assume that there are precedence constraints for the activities. The precedence constraint  $A \ll B$  specifies that activity  $A$  must not finish later than activity  $B$  starts. The precedence constraints describe a partial order between the activities. The goal of scheduling is to decide a total order that satisfies (extends) the partial order (this corresponds to the definition of a unary resource) in such a way that each activity is scheduled within its time window. Last but not least we allow some activities to be so called *optional*. It means that it is not known in advance whether such activities are allocated to the resource or not. If the optional activity is allocated to the resource, that is, it is included in the final resource schedule then we call this activity *valid*. If the activity is known not to be allocated to the resource then we call the activity *invalid*. In other cases, that is, the activity is not decided to be or not to be allocated to the resource, we call the activity *undecided*. Optional activities are useful for modelling alternative resources for the activities (an optional activity is used for each alternative resource and exactly one optional activity becomes valid) or for modelling alternative processes to accomplish a job (each process may consist of a different set of activities).

Note that for the above defined problem of scheduling with time windows it is known that deciding about an existence of a feasible schedule is NP-hard in the strong sense (Garey & Johnson, 1979) even when no precedence relations or optional activities are considered, so there is a little hope even for a pseudo-polynomial solving algorithm. Hence using propagation rules and constraint satisfaction techniques is justified there.

## 3. CONSTRAINTS AND CONSTRAINT SERVICES

*Constraint satisfaction problem* is defined as a triple  $(X, D, C)$ , where  $X$  is a finite set of variables,  $D$  is a set of domains for these variables, each variable may have its own domain which is a finite set of values, and  $C$  is a set of constraints restricting possible combinations of the values assigned to variables (a constraint is a relation over the

variables' domains). The task is to find a value for each variable from the corresponding domain in such a way that all the constraints are satisfied (Dechter, 2003).

There exist many constraint solvers that provide tools for solving constraint satisfaction problems, for example ILOG Solver, Mozart or the clpfd library of SICStus Prolog. These solvers are typically based on combination of domain filtering with depth-first search. Domain filtering is a process of removing values from the domains that do not satisfy some constraint. Each constraint has a filtering algorithm assigned to it that does this job for the constraint, and these algorithms communicate via the domains of the variables – if a filtering algorithm shrinks a domain of some variable, the algorithms for constraints that use this variable propagate the change to other variables until a fixed point is reached or until some domain becomes empty. Such a procedure is called a (generalised) arc consistency. When all domains are reduced to singletons then the solution is found. If some domain becomes empty then no solution exists. In all other cases the search procedure splits the space of possible assignments by adding a new constraint (for example by assigning a value to the variable) and the solution is being searched for in sub-spaces defined by the constraint and its negation (other branching schemes may also be applied).

The constraint solvers usually provide an interface for user-defined filtering algorithms so the users may extend the capabilities of the solvers by writing their own filtering algorithms (Schulte, 2002). This interface consists of two parts: triggers and propagators. The user should specify when the filtering algorithm is called – a *trigger*. This is typically a change of domain of some variable, for example when the lower bound of the domain is increased, the upper bound is decreased, or any element is deleted from the domain. The *propagator* then describes how this change is propagated to domains of other variables. The constraint solver provides procedures for access to domains of variables and for operations over the domains (membership, union, intersection, etc.). The output of the propagator is a proposal how to change domains of other variables in the constraint. The algorithm may also deduce that the constraint cannot be satisfied (fail) or that the constraint is entailed (exit). We will describe the propagation rules in such a way that they can be easily transformed into a filtering algorithm in the above sense. Each propagation rule will consist of a trigger describing when the rule is activated and a propagator describing how the domains of other variables are changed.

## 4. RULES FOR THE PRECEDENCE GRAPH

As we mentioned above, precedence relations are defined among the activities. These precedence relations define a precedence graph which is an acyclic directed graph where nodes correspond to activities and there is an arc from A to B if  $A \ll B$ . Frequently, the scheduling algorithms need to know whether A must be before B in the schedule, that is whether there is a path from A to B in the precedence graph. It is possible to look for the path each time such a query occurs. However, if such queries occur frequently then it is more efficient to provide the answer immediately, that is, in time  $O(1)$ . This can be achieved by keeping a transitive closure of the precedence graph.

**Definition 1:** We say that a precedence graph G is *transitively closed* if for any path from A to B in G there is also an arc from A to B in G.

Defining the transitive closure is more complicated when optional activities are assumed. In particular, if  $A \ll B$  and  $B \ll C$  and B is undecided then we cannot deduce that  $A \ll C$  simply because if B is removed – becomes invalid – then the path from A to C is lost. Therefore, we need to define transitive closure more carefully.

**Definition 2:** We say that a precedence graph G with optional activities is *transitively closed* if for any two arcs A to B and B to C such that B is a valid activity and A and C are either valid or undecided activities there is also an arc A to C in G.

It is easy to prove that if there is a path from A to B such that A and B are either valid or undecided and all inner nodes in the path are valid then there is also an arc from A to B in a transitively closed graph (by induction of the path length). Hence, if no optional activity is used (activities are valid) then Definition 2 is identical to Definition 1.

In the next paragraphs we will propose a constraint model for the precedence graph and two propagation rules that maintain the transitive closure of the graph with optional activities. We index each activity by a unique number from the set  $1, \dots, n$ , where n is the number of activities. For each activity we use a 0/1 variable Valid indicating whether the activity is valid (1) or invalid (0). If the activity is not known yet to be valid or invalid then the domain of Valid is  $\{0, 1\}$ . The precedence graph is encoded in two sets attached to each activity. CanBeBefore is a set of indices of activities that can be before a given activity. CanBeAfter is a set of indices of activities that can be after the activity. If we add an arc between A and B ( $A \ll B$ ) then we remove the index of A from CanBeAfter(B) and the index of B from CanBeBefore(A). For simplicity reasons we will write A instead of the index of A. Note that these sets can be easily implemented as finite domains of two variables so a special data structure is not necessary. For this implementation we propose to include value 0 in above two sets to ensure that the domain is not empty even if the

activity is first or last (an empty domain in CSP indicates the non-existence of a solution). The value 0 is not assumed as an index of any activity in the propagation rules. To simplify description of propagation rules we define for every activity A the following sets (not kept in memory but computed on demand):

$$\begin{aligned} \text{MustBeAfter}(A) &= \text{CanBeAfter}(A) \setminus \text{CanBeBefore}(A) \\ \text{MustBeBefore}(A) &= \text{CanBeBefore}(A) \setminus \text{CanBeAfter}(A) \\ \text{Unknown}(A) &= \text{CanBeBefore}(A) \cap \text{CanBeAfter}(A). \end{aligned}$$

$\text{MustBeAfter}(A)$  and  $\text{MustBeBefore}(A)$  are sets of those activities that must be after and before the given activity A respectively.  $\text{Unknown}(A)$  is a set of activities that are not yet known to be before or after activity A.

We initiate the precedence graph in the following way. First, the variables  $\text{Valid}(A)$ ,  $\text{CanBeBefore}(A)$ , and  $\text{CanBeAfter}(A)$  with their domains are created for every activity A. Then the known precedence relations are added in the above-described way (domains of  $\text{CanBeBefore}(A)$  and  $\text{CanBeAfter}(A)$  are pruned). Finally, the  $\text{Valid}(A)$  variable for every valid activity A is set to 1 (activities that are known to be invalid from the beginning may be omitted from the graph).

Propagation rule /1/ is invoked when the validity status of the activity becomes known. “Valid(A) is instantiated” is its trigger. The part after  $\rightarrow$  is a propagator describing pruning of domains. “exit” means that the constraint represented by the propagation rule is entailed so the propagator is not further invoked (its invocation does not cause further domain pruning). We will use the same notation in all rules.

```
Valid(A) is instantiated  $\rightarrow$  /1/
  if Valid(A) = 0 then
    for each B do /* disconnect A from B */
      CanBeBefore(B)  $\leftarrow$  CanBeBefore(B)  $\setminus$  {A}
      CanBeAfter(B)  $\leftarrow$  CanBeAfter(B)  $\setminus$  {A}
    else /* Valid(A)=1 */
      for each B  $\in$  MustBeBefore(A) do
        for each C  $\in$  MustBeAfter(A)  $\setminus$  MustBeAfter(B) do
          /* new precedence B  $\ll$  C */
          CanBeAfter(C)  $\leftarrow$  CanBeAfter(C)  $\setminus$  {B}
          CanBeBefore(B)  $\leftarrow$  CanBeBefore(B)  $\setminus$  {C}
          if B  $\notin$  CanBeBefore(C) then // break the cycle
            post_constraint(Valid(B)=0  $\vee$  Valid(C)=0)
  exit
```

**Observation:** Note that rule /1/ maintains symmetry for all valid and undecided activities because the domains are pruned symmetrically in pairs. This symmetry can be defined as follows: if  $\text{Valid}(B) \neq 0$  and  $\text{Valid}(C) \neq 0$  then  $B \in \text{CanBeBefore}(C)$  if and only if  $C \in \text{CanBeAfter}(B)$ . This moreover implies that  $B \in \text{MustBeBefore}(C)$  if and only if  $C \in \text{MustBeAfter}(B)$ .

We shall show now, that if the entire precedence graph is known in advance (no arcs are added during the solving procedure), then rule /1/ is sufficient for keeping the (generalised) transitive closure according to Definition 2. To give a formal proof we need to define several notions more precisely.

Let  $J = \{0, 1, \dots, n\}$  be the set of activities, where 0 is a dummy activity with the sole purpose to keep all sets  $\text{CanBeAfter}(i)$  and  $\text{CanBeBefore}(i)$  nonempty for all  $1 \leq i \leq n$ . Furthermore, let  $G = (J \setminus \{0\}, E)$  be the given precedence graph on the set of activities, and  $G^T = (J \setminus \{0\}, T)$  its (generalised) transitive closure (note that the previously used notation  $i \ll j$  does not distinguish between the arcs which are given as input and those deduced by transitivity). The formal definition of the set T can be now given as follows:

1. if  $(i, j) \in E$  then  $(i, j) \in T$
2. if  $(i, j) \in T$  and  $(j, k) \in T$  and  $\text{Valid}(i) \neq 0$  and  $\text{Valid}(j) = 1$  and  $\text{Valid}(k) \neq 0$  then  $(i, k) \in T$

Furthermore, the set T is not maintained as a list of pairs of activities. Instead, it is represented using the set variables  $\text{CanBeAfter}(i)$  and  $\text{CanBeBefore}(i)$ ,  $1 \leq i \leq n$  in the following manner:  $(i, j) \in T$  if and only if  $i \notin \text{CanBeAfter}(j)$  and  $j \notin \text{CanBeBefore}(i)$ . The incremental construction of the set T can be described as follows.

Initialization: for every  $i \in J \setminus \{0\}$  set

- $\text{CanBeAfter}(i) \leftarrow J \setminus \{i\}$
- $\text{CanBeBefore}(i) \leftarrow J \setminus \{i\}$
- $\text{Valid}(i) \leftarrow \{0,1\}$

Set-up: for every arc  $(i,j) \in E$  set

- $\text{CanBeAfter}(j) \leftarrow \text{CanBeAfter}(j) \setminus \{i\}$
- $\text{CanBeBefore}(i) \leftarrow \text{CanBeBefore}(i) \setminus \{j\}$

Propagation: whenever an activity is made valid, call rule /1/

Clearly,  $T$  is empty after the initialization and  $T=E$  after the set-up. Now we are ready to state and prove formally that rule /1/ is sufficient for maintaining the set  $T$ .

**Proposition 1:** Let  $i_0, i_1, \dots, i_m$  be a path in  $E$  such that  $\text{Valid}(i_j)=1$  for all  $1 \leq j \leq m-1$  and  $\text{Valid}(i_0) \neq 0$  and  $\text{Valid}(i_m) \neq 0$  (that is, the endpoints of the path are both either valid or undecided and all inner points of the path are valid). Then  $(i_0, i_m) \in T$ , that is,  $i_0 \notin \text{CanBeAfter}(i_m)$  and  $i_m \notin \text{CanBeBefore}(i_0)$ .

**Proof:** We shall proceed by induction on  $m$ . The base case  $m=1$  is trivially true after the set-up. For the induction step let us assume that the statement of the lemma holds for all paths (satisfying the assumptions of the lemma) of length at most  $m-1$ . Let  $1 \leq j \leq m-1$  be an index such that  $\text{Valid}(i_j) \leftarrow 1$  was set last among all inner points  $i_1, \dots, i_{m-1}$  on the path. By the induction hypothesis we get

- $i_0 \notin \text{CanBeAfter}(i_j)$  and  $i_j \notin \text{CanBeBefore}(i_0)$  using the path  $i_0, \dots, i_j$
- $i_j \notin \text{CanBeAfter}(i_m)$  and  $i_m \notin \text{CanBeBefore}(i_j)$  using the path  $i_j, \dots, i_m$

We shall distinguish two cases. If  $i_m \in \text{MustBeAfter}(i_0)$  (and thus by symmetry also  $i_0 \in \text{MustBeBefore}(i_m)$ ) then by definition  $i_m \notin \text{CanBeBefore}(i_0)$  and  $i_0 \notin \text{CanBeAfter}(i_m)$  and so the claim is true trivially. Thus let us in the remainder of the proof assume that  $i_m \notin \text{MustBeAfter}(i_0)$ .

Now let us show that  $i_0 \in \text{CanBeBefore}(i_j)$  must hold, which in turn (together with  $i_0 \notin \text{CanBeAfter}(i_j)$ ) implies  $i_0 \in \text{MustBeBefore}(i_j)$ . Let us assume by contradiction that  $i_0 \notin \text{CanBeBefore}(i_j)$ . However, at the time when both  $i_0 \notin \text{CanBeAfter}(i_j)$  and  $i_0 \notin \text{CanBeBefore}(i_j)$  became true, that is, when the second of these conditions was made satisfied by rule /1/, rule /1/ must have posted the constraint  $(\text{Valid}(i_0)=0 \vee \text{Valid}(i_j)=0)$  which contradicts the assumptions of the lemma. By a symmetric argument we can prove that  $i_m \in \text{MustBeAfter}(i_j)$ . Thus when rule /1/ is triggered by setting  $\text{Valid}(i_j) \leftarrow 1$  both  $i_0 \in \text{MustBeBefore}(i_j)$  and  $i_m \in \text{MustBeAfter}(i_j)$  hold (and  $i_m \notin \text{MustBeAfter}(i_0)$  is assumed), and therefore rule /1/ removes  $i_m$  from the set  $\text{CanBeBefore}(i_0)$  as well as  $i_0$  from the set  $\text{CanBeAfter}(i_m)$ , which finishes the proof.

From now on there will be no need to distinguish between the “original” arcs from  $E$  and the transitively deduced ones, so we will work solely with the set  $T$ . To simplify notation we shall switch back to the  $A \ll B$  notation (which is equivalent to  $(A, B) \in T$ ).

In some situations arcs may be added to the precedence graph during the solving procedure, either by the user, by the scheduler, or by other filtering algorithms like the one described in the next section. The following rule updates the precedence graph to keep transitive closure when an arc is added to the precedence graph.

```

A ≪ B is added → /2/
  CanBeAfter(B) ← CanBeAfter(B) \ {A}
  CanBeBefore(A) ← CanBeBefore(A) \ {B}
  if A ≪ CanBeBefore(B) then // break the cycle
    post_constraint(Valid(A)=0 ∨ Valid(B)=0)
  else
    if Valid(A)=1 then // transitive closure
      for each C ∈ MustBeBefore(A) \ MustBeBefore(B) do
        add C ≪ B
    if Valid(B)=1 then // transitive closure
      for each C ∈ MustBeAfter(B) \ MustBeAfter(A) do
        add A ≪ C
  exit

```

The rule /2/ does the following. If a new arc  $A \ll B$  is added then the sets  $\text{CanBeBefore}(A)$  and  $\text{CanBeAfter}(B)$  are updated. If a cycle is detected then the cycle is broken in the same way as in rule /1/. The rest of the propagation rule ensures that if an arc is added and one of its endpoints is valid then other arcs are added recursively to keep a transitive closure. The following proposition shows that all necessary arcs are added by rule /2/.

**Proposition 2:** If the precedence graph  $G$  is transitively closed and arc  $A \ll B$  is added to  $G$  then the propagation rule /2/ updates the precedence graph  $G$  to be transitively closed again.

**Proof:** Assume that arc  $A \ll B$  is added into  $G$  at a moment when arc  $B \ll C$  is already present in  $G$ . Moreover assume that  $\text{Valid}(A) \neq 0$ ,  $\text{Valid}(B) = 1$ , and  $\text{Valid}(C) \neq 0$ . We want to show that  $A \ll C$  is in  $G$  after rule /2/ is fired by the addition of  $A \ll B$ . The presence of arc  $B \ll C$  implies that  $C \in \text{MustBeAfter}(B)$  (and by symmetry also  $B \in \text{MustBeBefore}(C)$ ). Now there are two possibilities. Either  $C \notin \text{MustBeAfter}(A)$  in which case rule /2/ adds the arc  $A \ll C$  into  $G$ , or  $C \in \text{MustBeAfter}(A)$  (and by symmetry also  $A \in \text{MustBeBefore}(C)$ ) which means that arc  $A \ll C$  was already present in  $G$  when arc  $A \ll B$  was added.

The case when arc  $A \ll B$  is added into  $G$  at a moment when arc  $C \ll A$  is already present in  $G$  and  $\text{Valid}(C) \neq 0$ ,  $\text{Valid}(A) = 1$ ,  $\text{Valid}(B) \neq 0$  holds can be handled similarly.

Thus when an arc is added into  $G$ , all paths of length two which include this new arc are either already spanned by a transitive arc, or the transitive arc is added by rule /2/. In the latter case this may invoke adding more and more arcs. However, this process is obviously finite (cannot cycle) as an arc is added into  $G$  only if it is not present in  $G$ , and if an arc is removed from  $G$  (breaking the cycle), it can never be added back as one of its endpoints becomes invalid (and thus is permanently disconnected from  $G$ ).

Therefore, it is easy to see, that when the process of recursive arc additions terminates, the graph  $G$  is transitively closed. Indeed, for every path of length two in  $G$  one of the arcs is added later than the other, and we have already seen that at a moment of such an addition the transitive arc is either already on  $G$  or is added by rule /2/ in the next step.

## 5. RULES FOR TIME WINDOWS

An absolute position of the activity in time is frequently restricted by a *release time* and *deadline* that define a *time window* for processing the activity. The activity cannot start before the release time and it must be finished before the deadline. We assume the activity to be uninterruptible so it occupies the resource from its start till its completion. The processing time of activity  $A$  is constant, we denote it by  $p(A)$ . The goal of time window filtering is to remove time points from the time window when the activity cannot be processed. Usually, only the lower and upper bounds of the time window change so we are speaking about shrinking the time window.

The standard constraint model for time allocation of the activity assumes two variables –  $\text{start}(A)$  and  $\text{end}(A)$  – describing when the activity  $A$  starts and completes. Initially, the domain for the variable  $\text{start}(A)$  is  $[\text{release\_time}(A), \text{deadline}(A) - p(A)]$  and, similarly, the initial domain for the variable  $\text{end}(A)$  is  $[\text{release\_time}(A) + p(A), \text{deadline}(A)]$ . If these two initial domains are empty then the activity is made invalid. We will use the following notation to describe bounds of the above domains:

$$\begin{aligned} \text{est}(A) &= \min(\text{start}(A)) && \text{earliest start time} \\ \text{lst}(A) &= \max(\text{start}(A)) && \text{latest start time} \\ \text{ect}(A) &= \min(\text{end}(A)) && \text{earliest completion time} \\ \text{lct}(A) &= \max(\text{end}(A)) && \text{latest completion time} \end{aligned}$$

This notation can be extended in a natural way to sets of activities. Let  $\Omega$  be a set of activities, then:

$$\begin{aligned} \text{est}(\Omega) &= \min\{\text{est}(A), A \in \Omega\} \\ \text{lst}(\Omega) &= \max\{\text{lst}(A), A \in \Omega\} \\ \text{ect}(\Omega) &= \min\{\text{ect}(A), A \in \Omega\} \\ \text{lct}(\Omega) &= \max\{\text{lct}(A), A \in \Omega\} \\ p(\Omega) &= \sum\{p(A), A \in \Omega\} \end{aligned}$$

During propagation, we will be increasing  $\text{est}$  and decreasing  $\text{lct}$  which corresponds to shrinking the time window for the activity. For simplicity reasons we use a formula  $\text{est}(A) \leftarrow X$  to describe a requested change of  $\text{est}(A)$  which actually means  $\text{est}(A) \leftarrow \max(\text{est}(A), X)$ . Similarly  $\text{lct}(A) \leftarrow X$  means  $\text{lct}(A) \leftarrow \min(\text{lct}(A), X)$ .

The time windows can be used to deduce a new precedence between activities. In particular, if  $\text{est}(A) + p(A) + p(B) > \text{lct}(B)$  then activity  $A$  cannot be processed before activity  $B$  and hence we can deduce  $B \ll A$ . This is called a *detectable precedence* in (Vilím, 2002). Vice versa, the precedence graph can be used to shrink time windows of the activities. In particular, we can compute the earliest completion time of the set of valid activities that must be processed before some activity  $A$  and the latest start time of the set of valid activities that must be processed after  $A$ . These two numbers define bounds of the time window for  $A$ . Formally:

$$\begin{aligned} \text{est}(A) &\leftarrow \max \{ \text{est}(\Omega) + p(\Omega) \mid \Omega \subseteq \{X \mid X \ll A \text{ \& Valid}(X)=1\} \} \\ \text{lct}(A) &\leftarrow \min \{ \text{lct}(\Omega) - p(\Omega) \mid \Omega \subseteq \{X \mid A \ll X \text{ \& Valid}(X)=1\} \} \end{aligned}$$

The above two formulas are special cases of the *energy precedence constraint* (Laborie, 2003) for unary resources. Note also that the new bound for  $\text{est}(A)$  can be computed in  $O(n \cdot \log n)$  time, where  $n$  is the number of activities in  $\Theta = \{X \mid X \ll A \text{ \& Valid}(X)=1\}$ , rather than exploring all subsets  $\Omega \subseteq \Theta$ . The algorithm is based on the following observation: if  $\Omega'$  is the set with the maximal  $\text{est}(\Omega') + p(\Omega')$  then  $\Omega' \supseteq \{X \mid X \in \Theta \text{ \& } \text{est}(\Omega') \leq \text{est}(X)\}$ , otherwise adding such  $X$  to  $\Omega'$  will increase  $\text{est}(\Omega') + p(\Omega')$ . Consequently, it is enough to explore sets  $\Omega_X = \{Y \mid Y \in \Theta \text{ \& } \text{est}(X) \leq \text{est}(Y)\}$  for each  $X \in \Theta$  which is done by the following algorithm (the new bound is computed in the variable *end*):

```

dur ← 0
end ← inf
for each  $Y \in \{X \mid X \ll A \text{ \& Valid}(X)=1\}$  in non-increasing order of  $\text{est}(Y)$  do
    dur ← dur + p(Y)
    end ← max(end, est(Y) + dur)

```

The bound for  $\text{lct}(A)$  can be computed in a symmetrical way in  $O(n \cdot \log n)$  time, where  $n$  is the number of activities in  $\{X \mid A \ll X \text{ \& Valid}(X)=1\}$ .

We now present two groups of propagation rules working with time windows and a precedence graph. The first group of rules realise the energy precedence constraint in an incremental way by reacting to changes in the precedence graph. The rules are invoked by making the activity valid /1a/ and by adding a new precedence relation /2a/. Because these rules have the same triggers as the rules for the precedence graph, they can be actually combined with them. Hence, we name the new rules using the number of the corresponding rule for the precedence graph.

The rules shrink the time windows using information about the precedence relations as described above. Only valid activities influence time windows of other (non invalid) activities. This corresponds to our requirement that optional activities that are not yet known to be valid should not influence other activities but they can be influenced. Notice also that if  $A \ll C$ ,  $C \ll B$ , and  $C$  is valid then it is enough to explore possible increase of  $\text{est}(C)$  only. The reason is that if  $\text{est}(C)$  is really increased then the rule /3/ is invoked for  $C$  (see below) and the change is propagated directly to  $\text{est}(B)$ . Similarly, only activities  $B$  such that there is no valid activity  $C$  in between  $B$  and  $A$  are explored for change of  $\text{lct}(B)$ .

When activity  $A$  becomes valid and  $B$  is after  $A$  ( $A \ll B$ ) or when  $A$  is valid and arc  $A \ll B$  is added then  $A$  can (newly) participate in sets  $\Omega_X$  that are used to compute  $\text{est}$  of  $B$  (see above). Visibly, only sets containing  $A$  are of interest because only these sets can lead to change of  $\text{est}(B)$ . The other sets  $\Omega_X$  used to update  $\text{est}(B)$  have already been explored or will be explored when calling the rules for some valid activity in  $\Omega_X$ . Moreover, all valid activities  $C$  such that  $C \ll A$  are used to compute  $\text{est}(A)$  so they can complete together no later than in  $\text{est}(A)$ . Hence these activities do not influence directly  $\text{est}(B)$  (they influence it through changes of  $\text{est}(A)$ ). Thus, we need to explore all subsets of valid activities  $X$  such that  $X \ll B$  and  $\neg X \ll A$  and these subsets contain  $A$ . Only these subsets can deduce a possible change of  $\text{est}(B)$ . These are exactly the sets used in rules /1a/ and /2a/. A symmetrical analysis can be done for activities  $B$  before  $A$ . Note also that sets  $\Omega'$  in rules /1a/ and /2a/ can be explored in the same way as we described for the energy precedence constraint above.

```

Valid(A) is instantiated → /1a/
if Valid(A)=1 then
    for each  $B \in \text{MustBeAfter}(A)$  such that  $\neg \exists C \text{ Valid}(C)=1 \text{ \& } A \ll C \text{ \& } C \ll B$  do
        let  $\Omega = \{X \mid X \in \text{MustBeBefore}(B) \text{ \& } X \in \text{CanBeAfter}(A) \text{ \& Valid}(X)=1\}$ 
         $\text{est}(B) \leftarrow \max \{ \text{est}(\Omega' \cup \{A\}) + p(\Omega') + p(A) \mid \Omega' \subseteq \Omega \}$ 
    for each  $B \in \text{MustBeBefore}(A)$  such that  $\neg \exists C \text{ Valid}(C)=1 \text{ \& } B \ll C \text{ \& } C \ll A$  do
        let  $\Omega = \{X \mid X \in \text{MustBeAfter}(B) \text{ \& } X \in \text{CanBeBefore}(A) \text{ \& Valid}(X)=1\}$ 
         $\text{lct}(B) \leftarrow \min \{ \text{lct}(\Omega' \cup \{A\}) - p(\Omega') - p(A) \mid \Omega' \subseteq \Omega \}$ 
exit

```

/2a/

```

A«B is added →
  if Valid(A)=1 & Valid(B)≠0 then
    let  $\Omega = \{X \mid X \in \text{MustBeBefore}(B) \ \& \ X \in \text{CanBeAfter}(A) \ \& \ \text{Valid}(X)=1\}$ 
     $\text{est}(B) \leftarrow \max\{\text{est}(\Omega' \cup \{A\}) + p(\Omega') + p(A) \mid \Omega' \subseteq \Omega\}$ 
  if Valid(B)=1 & Valid(A)≠0 then
    let  $\Omega = \{X \mid X \in \text{MustBeAfter}(A) \ \& \ X \in \text{CanBeBefore}(B) \ \& \ \text{Valid}(X)=1\}$ 
     $\text{lct}(A) \leftarrow \min\{\text{lct}(\Omega' \cup \{B\}) - p(\Omega') - p(B) \mid \Omega' \subseteq \Omega\}$ 
exit

```

The second group of rules is triggered by shrinking the time window (/3/ for increased est and /4/ for decreased lct). The rules in this group can deduce that the activity is invalid, if it has an empty time window, and they can deduce a new detectable precedence. Moreover, if the activity is valid then the change of its time window is propagated to other activities whose relative position to a given activity is known (they are before or after the given activity). If est of valid activity A is increased then it may influence est of B such that  $A \ll B$  (note that B is either valid or undecided, because invalid activities are disconnected from the graph). This happens if and only if  $\text{est}(B) \leq \text{est}(\Omega_A) + p(\Omega_A)$  (see above for the definition of  $\Omega_A$  with respect to B). Notice that rule /3/ computes  $\text{est}(\Omega_A) + p(\Omega_A)$  to update  $\text{est}(B)$ . Symmetrically, rule /4/ updates  $\text{lct}(B)$  for activities B such that  $B \ll A$ , if necessary. Hence, the propagation rules incrementally maintain the energy precedence constraint.

/3/

```

est(A) is increased →
  if Valid(A)=0 or  $\text{est}(A) + p(A) > \text{lct}(A)$  then
    Valid(A) ← 0
    exit
  else
     $\text{ect}(A) \leftarrow \text{est}(A) + p(A)$ 
    for each  $B \in \text{Unknown}(A)$  do
      if  $\text{est}(A) + p(A) + p(B) > \text{lct}(B)$  then
         $B \ll A$  /* detectable precedence */
    if Valid(A)=1 then
      for each  $B \in \text{MustBeAfter}(A)$  such that  $\neg \exists C \text{ Valid}(C)=1 \ \& \ A \ll C \ \& \ C \ll B$  do
         $\text{est}(B) \leftarrow \text{est}(A) + p(A) + \sum\{p(X) \mid X \in \text{MustBeBefore}(B) \ \& \ \text{est}(A) \leq \text{est}(X) \ \& \ \text{Valid}(X)=1\}$ 

```

/4/

```

lct(A) is decreased →
  if Valid(A)=0 or  $\text{est}(A) + p(A) > \text{lct}(A)$  then
    Valid(A) ← 0
    exit
  else
     $\text{lct}(A) \leftarrow \text{lct}(A) - p(A)$ 
    for each  $B \in \text{Unknown}(A)$  do
      if  $\text{est}(B) + p(B) + p(A) > \text{lct}(A)$  then
         $A \ll B$  /* detectable precedence */
    if Valid(A)=1 then
      for each  $B \in \text{MustBeBefore}(A)$  such that  $\neg \exists C \text{ Valid}(C)=1 \ \& \ B \ll C \ \& \ C \ll A$  do
         $\text{lct}(B) \leftarrow \text{lct}(A) - p(A) - \sum\{p(X) \mid X \in \text{MustBeAfter}(B) \ \& \ \text{lct}(X) \leq \text{lct}(A) \ \& \ \text{Valid}(X)=1\}$ 

```

## 6. CONCLUSIONS

The paper reports a work in progress on constraint models for the unary resource with precedence relations between the activities and time windows for the activities. Optional activities that may or may not be allocated to the resource are also assumed. We propose a set of propagation rules that keep a transitive closure of the precedence relations, deduce additional precedence constraints based on time windows, and shrink the time windows for the activities. These rules are intended to complement the existing filtering algorithms based on edge-finding etc. to further improve domain pruning. Our next steps include formal complexity analysis, detail comparison to existing propagation rules (edge finder, etc.), implementation of the proposed rules, and testing in real-life environment.



## ACKNOWLEDGEMENTS

The research is supported by the Czech Science Foundation under the contract no. 201/04/1102. We would like to thank anonymous reviewers for comments on early draft.

## REFERENCES

- Baptiste, P. and Le Pape, C. 1996. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling, *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group (PLANSIG)*.
- Baptiste P., Le Pape C., and Nuijten W. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publishers.
- Brucker P. 2001. *Scheduling Algorithms*, Springer Verlag.
- Cesta A. and Stella C. 1997. A Time and Resource Problem for Planning Architectures, *Recent Advances in AI Planning (ECP'97)*, LNAI 1348, Springer Verlag, 117-129.
- Dechter R. 2003. *Constraint Processing*, Morgan Kaufmann.
- Focacci F., Laborie P., and Nuijten W. 2000. Solving Scheduling Problems with Setup Times and Alternative Resources, *Proceedings of AIPS 2000*.
- Garey M. R. and Johnson D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Company, San Francisco.
- Laborie P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, *Artificial Intelligence*, 143, 151-188.
- Schulte C. 2002. *Programming Constraint Services, High-Level Programming of Standard and New Constraint Services*, Springer Verlag.
- Torres P. and Lopez P. 1999. On Not-First/Not-Last conditions in disjunctive scheduling, *European Journal of Operational Research*, 127, 332-343.
- Vilím P. 2002. Batch Processing with Sequence Dependent Setup Times: New Results, *Proceedings of the 4th Workshop of Constraint Programming for Decision and Control, CPDC'02*, Gliwice, Poland.
- Vilím P., Barták R., and Čepěk O. 2004. Unary Resource Constraint with Optional Activities, *Principles and Practice of Constraint Programming (CP 2004)*, LNCS 3258, Springer Verlag, 62-76.

# **Theoretical Scheduling**

Full Papers

# SCHEDULING ON A SINGLE MACHINE TO MINIMIZE TOTAL FLOW TIME WITH JOB REJECTIONS

David P. Bunde\*

*University of Illinois at Urbana-Champaign*

bunde@uiuc.edu

**Abstract** We consider the problem of minimizing flow time on a single machine supporting preemption that can reject jobs at a cost. Even if all jobs have the same rejection cost, we show that no online algorithm can have competitive ratio better than  $(2 + \sqrt{2})/2 \approx 1.707$  in general or  $e/(e - 1) \approx 1.582$  if all jobs are known to have the same processing time. We also give an optimal offline algorithm for unit-length jobs with arbitrary rejection costs. This leads to a pair of 2-competitive online algorithms for unit-length jobs, one when all rejection costs are equal and one when they are arbitrary. Finally, we show that the offline problem is NP-hard even when each job's rejection cost is proportional to its processing time.

**Keywords:** Complexity of Scheduling Problems, Theoretical Scheduling

## 1 Introduction

In classical scheduling problems, the input is a stream of jobs, all of which must be completed. This type of problem is just one aspect of planning for industrial production because it ignores the process of deciding what jobs to complete. One view of industrial planning divides it into four levels, motivated by semiconductor manufacturing [9]. At the top level, management decides which product lines to promote and how much production to devote to each. The second level decides on the desired production for specific products and the third converts this into requests for components that must be completed by specific times. At the bottom level is a classical scheduling problem involving the assignment of tasks to specific machinery. Classical scheduling ignores the decisions at the top levels since they do not fit neatly into a mathematical framework, but these decisions cannot be made without some assurance that the resulting goals are feasible, requiring input from the scheduler. Although interaction between levels may be limited in many cases, it would be advantageous to combine the job selection and scheduling decisions.

In this paper, we consider one such combination. Our system receives requests that potentially exceed its capacity and focuses on a subset of them, satisfying this subset while rejecting the rest. More formally, we attempt to schedule  $n$  jobs on a single machine. Each job  $J_i$  completes after running exclusively on the machine for its *processing time*  $p_i$ , incurs a *rejection cost*  $c_i$  if it is rejected, and cannot be run or rejected before its *release time*  $r_i$ . The time at which algorithm  $A$  completes or rejects job  $J_i$  is its *completion time*, denoted  $C_i^A$ . The *flow time*  $F_i^A$  of job  $J_i$  is  $C_i^A - r_i$ , the time between  $J_i$ 's arrival and its completion or rejection. The *cost* of job  $J_i$ , denoted  $cost_i^A$ , is its flow time ( $F_i^A$ ) if algorithm  $A$  completes job  $J_i$  or its flow time plus its rejection cost ( $F_i^A + c_i$ ) if job  $J_i$  is rejected. The objective is to minimize  $\sum_{i=1}^n cost_i^A$ , the total cost. This paper considers machines supporting *preemption*, meaning jobs can be interrupted and resumed later without penalty. (This model is called *preempt-resume*.) We primarily consider the *online* problem, where the algorithm

\*Partially supported by NSF grant CCR 0093348.

has no knowledge of a job until its release time, but we also consider the *offline* problem, where all aspects of the input are known initially.

Although this problem differs from the factory setting described above, we believe it is an interesting way to add an element of planning to the scheduling decision. The total flow time metric accurately captures the quality of service for an interactive system. Bansal et al. [2] discuss minimizing total flow time with rejections in the context of managing a “to do” list. Throughout the day, you receive tasks from coworkers. Each comes with two (potentially unknown) parameters, the amount of work it will take you and the time it will take someone else. These correspond to the task’s processing time and its rejection cost, respectively. The time a coworker waits for a requested task is either your time to complete it, or the time you wait before delegating it to someone else plus their time to complete it. Minimizing the sum of coworker waiting times is exactly our metric.

**Related work.** Scheduling to minimize flow time without rejections has been extensively studied; see the survey by Pruhs et al. [10]. In the uniprocessor preemptive setting, the algorithm Shortest Remaining Processing Time (SRPT), which always runs the job with the least processing time remaining, is known to be optimal [12, 15]. In other settings, however, optimal solutions cannot always be found. When this occurs, one goal is to give algorithms with performance guarantees. We say that an online algorithm is *c-competitive* or an offline algorithm is a *c-approximation* if the cost of its solution is never more than  $c$  times the cost of the optimal solution.

Minimizing flow time with rejections has previously been considered by Bansal et al. [2], whose work is included in Bansal’s dissertation [1]. When the rejection costs are arbitrary, they show that no randomized algorithm is better than  $\sqrt[4]{n}$  or  $\sqrt{C}$ -competitive, where  $C$  is the ratio between the largest and smallest rejection costs. When all jobs have the same rejection cost, they give a 2-competitive algorithm and a lower bound of 1.5 that holds even when all jobs have processing time 1. Using ideas of Bansal and Dhandhere [3], they generalize the algorithm to the weighted problem by rounding weights to powers of 2. The result is  $O(\log^2 W)$ -competitive if all jobs in a weight class have the same rejection cost, where  $W$  is the ratio between the largest and smallest weights. They also give an algorithm for general rejection costs that is  $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ -competitive if it runs on a processor that is  $(1 + \epsilon)$  times faster than the optimal solution’s processor. Finally, they mention a  $(1 + \epsilon)$ -approximation with running time  $n^{O(\log n/\epsilon^2)}$  for the offline unweighted problem when all jobs have the same rejection cost.

Engels et al. [6] consider offline algorithms to minimize a related metric, weighted completion time ( $\sum_i w_i C_i^A + \text{cost of rejections}$ ). They show this problem is NP-complete even when all jobs arrive at time 0, have weight equal to their processing time, and have the same rejection cost. (This implies that minimizing weighted flow time is NP-hard under the same conditions.) They also give optimal algorithms for some special cases when all jobs arrive at time 0 on a single machine and approximation algorithms for scheduling multiple unrelated machines.

Scheduling with rejections was introduced by Bartal et al. [4], who tried to minimize makespan ( $\max_i C_i^A + \text{cost of rejections}$ ) on a nonpreemptive multiprocessor. Seiden [13] gives better bounds by allowing preemption and Hoogeveen [8] considers the offline problem with preemption. Sen-gupta [14] considers scheduling with rejections to minimize maximum lateness and tardiness.

**Our results.** We give both algorithms and lower bounds for minimizing flow time with rejections. Many of our results require the assumption that all processing times are the same and each arrival time is a multiple of this value. We call this special case *unit-length jobs* and assume the processing times are 1 and job arrival times are integers. This corresponds to an application with events occurring on clock ticks and/or extremely short jobs. Focusing on unit-length jobs allows us to assume that any job started is completed before another arrives. Our results are the following:

- When all jobs have the same rejection cost, we show that no algorithm has a constant competitive ratio below  $e/(e - 1) \approx 1.582$  if all jobs have unit length or  $(2 + \sqrt{2})/2 \approx 1.707$

if distinct processing times are allowed. The previous best lower bound in both cases was 1.5 [2].

- We give an optimal offline algorithm and a 2-competitive online algorithm for the special case of unit-length jobs with arbitrary rejection costs. Nothing was previously known in this case.
- We introduce a family of online algorithms and show that one member is 2-competitive for unit-length jobs, each having the same rejection cost. This result is not surprising since the algorithm above achieves the same competitive ratio with arbitrary rejection costs and the algorithm of Bansal et al. [2] achieves it with arbitrary processing times. We include it, however, because both the algorithm and proof technique are different.
- We show that the problem is NP-complete even when the rejection cost of a job is proportional to its processing time. This is the first offline hardness result for the unweighted problem, answering an open question posed by Bansal [1].

We also have, but defer to a later version of this paper, an optimal offline algorithm for the special case when all jobs have the same (non-unit) processing time.

The rest of this paper is organized as follows. Section 2 contains our lower bounds on the competitive ratio of online algorithms. Section 3 gives algorithms for unit-length jobs. Section 4 contains the NP-completeness proof. Finally, Section 5 concludes with a discussion of open problems.

## 2 Lower bounds

Now we give lower bounds on the competitive ratio when all jobs have the same rejection cost, which we denote with  $c$ . In our proofs, we use  $OPT$  to refer to the optimal algorithm and  $ALG$  to refer to a hypothetical algorithm whose competitive ratio is better than our bounds.

### 2.1 When all jobs have unit length

Our first lower bound applies to unit-length jobs. It is a generalization of one presented by Bansal et al. [2]. The instance in their bound has two parts. The first is a stream of jobs, one released at each unit time. The second is a single “extra” job, released at time 0. The stream continues until the algorithm rejects a job, which we may assume to be the extra one. Their lower bound of  $3/2$  comes from the tradeoff between rejecting the extra job quickly, in which case  $OPT$  finishes all jobs quickly and there is little to amortize the rejection cost against, and rejecting the extra job later, in which case  $OPT$  rejects it immediately and the algorithm’s flow time is too large.

Our lower bound instance consists of a stream of jobs arriving one per time unit and  $k$  extra jobs released at time 0. We will show that the algorithm cannot achieve a competitive ratio better than  $e/(e-1)$  because it must wait until a time  $\tau_{k,i}c$  to make the  $i^{\text{th}}$  rejection, but that delaying the rejections to these times yields the same competitive ratio. We define  $\tau_{k,i}$  recursively as follows:

$$\begin{aligned}\tau_{k,0} &= 0 \\ \tau_{k,i} &= \frac{i + \sum_{j=1}^{i-1} \tau_{k,j} - (i-1)\tau_{k,i}}{(k+1)/(e-1)}.\end{aligned}\tag{1}$$

Subtracting  $\tau_{k,i-1}$  from  $\tau_{k,i}$  and solving for the difference gives the following alternate definition:

$$\tau_{k,i} = \sum_{j=1}^i \frac{1}{(k+1)/(e-1) + (j-1)}\tag{2}$$

In this paper, we use the following observations about  $\tau_{k,k}$ :

LEMMA 1 For all values of  $k$ ,  $\tau_{k,k} < 1$ .

LEMMA 2  $\lim_{k \rightarrow \infty} \tau_{k,k} = 1$

These lemmas have technical and unenlightening proofs, which appear in the Appendix. Instead, we proceed with the lower bound:

THEOREM 3 No deterministic algorithm for flow time with rejections is  $(e/(e-1) - \epsilon)$ -competitive for any constant  $\epsilon > 0$  even when all jobs have unit length and the same rejection cost.

**Proof:** Suppose to the contrary that  $ALG$  is  $(e/(e-1) - \epsilon)$ -competitive. Choose  $k$  large enough so  $\tau_{k,k} \geq 1 - \epsilon/2$ , which is possible by Lemma 2. We consider the instance described above consisting of a stream of jobs, one released at each unit time, and  $k$  extra jobs released at time 0. All jobs have unit length and rejection cost  $c = 6k^2/\epsilon$ .

Let time  $t_i$  be when  $ALG$  makes its  $i^{\text{th}}$  rejection, assuming the stream continues until time  $t_i$ . We consider two cases. The first is  $t_i < \tau_{k,i}c$  for some smallest value  $i$ . In this case, the last job of the stream arrives at time  $t_i$ .  $ALG$  has cost at least  $ic + \sum_{j=1}^i t_j + (k-i+1)t_i + \binom{k+1-i}{2} > ic + \sum_{j=1}^i t_j + (k-i+1)t_i$ .  $OPT$  has cost at most  $(k+1)t_i + \binom{k+1}{2}$ . If  $i = 1$ , the competitive ratio is at least  $ec/((e-1)c + k^2) > e/(e-1) - \epsilon$  and we are done. If  $i > 1$ , the cost of  $OPT$  is at most  $(k+1)t_i + \binom{k+1}{2} \leq (k+1)t_i + k^2 \leq (k+1)t_i(1 + \epsilon/(3(e-1))) \leq (k+1)t_i(1 + \epsilon(e-1)/e)$ . Algebraic manipulation of the ratio between these bounds leads to a contradiction:

$$\begin{aligned} \frac{ic + \sum_{j=1}^i t_j + (k-i+1)t_i}{(k+1)t_i(1 + \epsilon(e-1)/e)} &= \frac{1}{1 + \epsilon(e-1)/e} + \frac{ic + \sum_{j=1}^{i-1} t_j - (i-1)t_i}{(k+1)t_i(1 + \epsilon(e-1)/e)} \\ &\geq \frac{1}{1 + \epsilon(e-1)/e} + \frac{ic + \sum_{j=1}^{i-1} \tau_{k,j}c - (i-1)\tau_{k,i}c}{(k+1)\tau_{k,i}c(1 + \epsilon(e-1)/e)} \\ &= \frac{e}{(e-1)(1 + \epsilon(e-1)/e)} \\ &> \frac{e}{e-1} - \epsilon. \end{aligned}$$

In the second case,  $t_i \geq \tau_{k,i}c$  for each  $i$ . In this case, the last job of the stream arrives at time  $t_k$ . The cost of  $ALG$  is  $kc + \sum_{i=1}^k t_i + t_k + 1 \geq kc + \sum_{i=1}^{k-1} \tau_{k,i}c + 2t_k + 1$ .  $OPT$  can immediately reject all  $k$  extra jobs, so its cost is at most  $kc + t_k + 1$ . Again, manipulation of the ratio between these bounds leads to a contradiction:

$$\begin{aligned} \frac{kc + \sum_{i=1}^k \tau_{k,i}c + t_k + 1}{kc + t_k + 1} &\geq \frac{kc + \sum_{i=1}^k \tau_{k,i}c + \tau_{k,k}c + 1}{kc + \tau_{k,k}c + 1} = 1 + \frac{\sum_{i=1}^k \tau_{k,i}}{kc + \tau_{k,k} + 1} \\ &> 1 + \frac{\sum_{i=1}^k \tau_{k,i}}{k+1+1/c} > 1 + \frac{\sum_{i=1}^k \tau_{k,i}}{(k+1)(1 + \epsilon/6)} \\ &\geq 1 + \frac{1}{(e-1)(1 + \epsilon/6)} - \frac{k\epsilon}{2(k+1)(1 + \epsilon/6)} \\ &\geq 1 + \frac{1 - \epsilon/3}{e-1} - \epsilon/2 \\ &> \frac{e}{e-1} - \epsilon. \end{aligned}$$

□

## 2.2 When jobs have arbitrary processing times

For jobs with arbitrary processing times, we prove the following stronger lower bound:

**THEOREM 4** *When jobs have uniform rejection cost and arbitrary processing time, no deterministic online algorithm has a constant competitive ratio better than  $(2 + \sqrt{2})/2 \approx 1.707$ .*

**Proof:** Let  $\gamma = (2 + \sqrt{2})/2$ . Suppose to the contrary that a deterministic algorithm  $ALG$  is  $(\gamma - \epsilon)$ -competitive. Without loss of generality, we assume  $1/\epsilon$  is an integer. Let  $c = 2\gamma/\epsilon$ .

Using these values, we give an input instance on which  $ALG$  has competitive ratio worse than  $\gamma - \epsilon$ . At time 0, release a job with processing time  $c/\gamma = 2/\epsilon$ . Starting at time  $c/\gamma - 1$ , release a job with processing time 1 every unit time until  $ALG$  rejects a job at some time  $t$ . If  $t < c/\gamma - 1$ , the instance has only one job and the competitive ratio is at least  $c/(c/\gamma) = \gamma > \gamma - \epsilon$ . Otherwise, the competitive ratio is

$$\frac{2t - (c/\gamma - 1) + c + 1}{\min\{t + 1, c\} + t + 1 - (c/\gamma - 1)}.$$

This is minimized when  $t = c - 1$ , so this ratio is at least

$$\frac{3c - c/\gamma}{2c - c/\gamma + 1}.$$

Since the ratio is at most 2, this is greater than

$$\frac{3c - c/\gamma - 2}{2c - c/\gamma} > \gamma - 2/c > \gamma - \epsilon.$$

□

Intuitively, the bound is improved by the flow time incurred before  $ALG$  rejects the long job.

## 3 Algorithms for unit-length jobs

Now we give algorithms for the special case of unit-length jobs. We describe these algorithms in terms of the *active* jobs at a particular time, those that have arrived but have not yet been completed or rejected.

### 3.1 Optimal offline algorithm

We begin by observing which jobs an offline optimal algorithm rejects. We call  $r_i + c_i$  the *rank* of job  $J_i$  and denote it with  $rank(J_i)$ .

**LEMMA 5** *If all jobs have unit processing time and  $OPT$  rejects job  $J_i$ , then no job with rank lower than  $rank(J_i)$  is active any time during the interval  $[r_i, r_i + c_i]$ .*

**Proof:** Suppose to the contrary that  $OPT$  rejects some job  $J_i$  but completes a job  $J_j$  during  $[r_i, r_i + c_i]$ , with  $r_j + c_j < r_i + c_i$ . Consider schedule  $OPT'$  that is identical to  $OPT$  except that  $OPT'$  rejects  $J_j$  at time  $r_j$  and runs  $J_i$  in the time freed up, i.e. starting at time  $C_j^{OPT} - 1$  and ending at  $C_j^{OPT}$ . Jobs  $J_i$  and  $J_j$  contribute  $c_i + C_j^{OPT} - r_j$  to  $OPT$  and  $C_i^{OPT'} - r_i + c_j$  to  $OPT'$ . By construction,  $C_i^{OPT'} = C_j^{OPT}$  and the cost of every job other than  $J_i$  and  $J_j$  is the same. Thus,  $cost(OPT') - cost(OPT) = -r_i + c_j - (c_i - r_j) = r_j + c_j - (r_i + c_i) < 0$ . Therefore, schedule  $OPT'$  is strictly better than  $OPT$ , a contradiction. □

Implicit in the proof of this lemma is that a rejected job can be swapped for a job of equal rank without affecting the total cost. Let a *normalized optimal solution* be an optimal solution that rejects the fewest number of jobs and always runs the active job with highest rank, breaking ties with the job number.

Lemma 5 leads us to the algorithm Offline Highest Rank First (*HRF-OFF*). At all times, *HRF-OFF* runs the active job with highest rank, again using job numbers to break ties. It rejects any job whose flow time equals its rejection cost, i.e., any job not completed by time equal to its rank. Each of *HRF-OFF*'s rejections occurs at the rejected job's arrival time.

**THEOREM 6** *Algorithm HRF-OFF is optimal for unit-length jobs with arbitrary rejection costs.*

**Proof:** Let *OPT* be a normalized optimal solution. By construction, *OPT* and *HRF-OFF* can only differ if they reject different jobs. Consider the first time this occurs. Since both algorithms always reject jobs when they arrive, this time is the arrival time  $r_i$  of some job  $J_i$ .

Suppose *HRF-OFF* rejects job  $J_i$  and *OPT* does not. This is the first difference between the schedules so *OPT* and *HRF-OFF* have the same active jobs just prior to time  $r_i$ . Also, Lemma 5 implies that *OPT* does not reject jobs ranked above  $J_i$  arriving between times  $r_i$  and  $C_i^{OPT}$ . Since both *OPT* and *HRF-OFF* run jobs in order of decreasing rank, they have the same higher-ranked active jobs throughout this interval. *HRF-OFF* is busy with higher-ranked jobs until time  $r_i + c_i$  because it rejects job  $J_i$ . Thus, the flow time of job  $J_i$  in *OPT* is greater than  $c_i$ , a contradiction.

Now suppose *OPT* rejects job  $J_i$  and *HRF-OFF* does not. Let  $t = C_i^{HRF-OFF} - 1$  be the time *HRF-OFF* starts job  $J_i$ . At time  $t$ , *HRF-OFF* has completed all jobs ranked above  $J_i$  that have arrived because it runs jobs in order of decreasing rank. (None are rejected since a job is rejected exactly when it cannot be completed by time equal to its rank and  $t < \text{rank}(J_i) \leq$  "higher" rank.) *OPT* also runs jobs in order of decreasing rank so it has also completed these jobs. By Lemma 5, *OPT* has no lower-ranked active jobs at time  $t$  since  $r_i \leq t < \text{rank}(J_i) = r_i + c_i$ . Thus, it has no active jobs at time  $t$  and running job  $J_i$  then instead of rejecting it is a valid schedule. This schedule has fewer rejections than *OPT* and no worse cost, a contradiction.  $\square$

### 3.2 2-competitive online algorithm

*HRF-OFF* makes most of its decisions online, but is an offline algorithm since running it online requires the ability to reject jobs retroactively. We call the algorithm that rejects jobs when their flow time equals their rejection cost Online Highest Rank First (*HRF-ON*) since it can run online.

By Theorem 6, *HRF-ON* is optimal if each rejection is made retroactively at the job's release time. Since each rejected job  $J_i$  accumulates flow time  $c_i$  before rejection, we get the following:

**THEOREM 7** *HRF-ON is 2-competitive for unit-length jobs having arbitrary rejection costs.*

In fact, *HRF-ON* is not  $(2 - \epsilon)$ -competitive for any constant  $\epsilon > 0$  even if we assume it rejects jobs once it is clear their flow time will exceed their rejection cost. Consider the following instance where all jobs have rejection cost  $c$ :  $\sqrt{c}$  jobs (the *main group*) arrive at time 0 and another job arrives at each time  $i$  for  $i = 0, \dots, c-1$  (the *stream*). The algorithm keeps all the main group jobs until time  $c - \sqrt{c}$ , at which point it rejects them one at a time, one per unit time. Thus, these jobs incur cost  $\sqrt{c}(c - \sqrt{c}) + \binom{\sqrt{c}}{2} + c\sqrt{c} = 2c^{3/2} - c/2 + \sqrt{c}/2$ . Jobs in the stream run as they arrive, for combined flow time and cost  $c$ . Thus, the total cost is  $2c^{3/2} + c/2 + \sqrt{c}/2$ . The optimal algorithm rejects the main group immediately and runs the stream as it arrives, for total cost  $c^{3/2} + c$ . Thus, the competitive ratio approaches 2 as  $c$  increases.

### 3.3 Online algorithm when all jobs have rejection cost $c$

Let the *pseudoschedule* of a schedule be the modification created by keeping each rejected job  $J_i$  active for  $c_i$  time units. Intuitively, these jobs wait until being instantly finished or rejected for free. Note that the pseudoschedule's total flow is the cost of the schedule from which it is made. When this schedule is an optimal schedule, we call the result an *optimal pseudoschedule*. The optimal pseudoschedule can be generated by an online simulation of *HRF-OFF* since *HRF-OFF* is optimal and decides whether to reject each job  $J_i$  within time  $c_i$  of its release.



Using this simulation, we define a family of algorithms called *ratio rejecting*. A member of this family is denoted  $RR(\rho)$  for a constant  $\rho > 1$ , though we omit  $\rho$  when discussing a general property of the family or when its value is clear from context. Let  $cost(EOPT, t)$  and  $cost(RR(\rho), t)$  be the costs incurred up to time  $t$  by the simulation and  $RR(\rho)$ , respectively.  $RR(\rho)$  always runs the most recently-arrived job. It rejects the active job with earliest arrival time whenever

$$\frac{cost(RR(\rho), t) + c}{cost(EOPT, t)} \leq \rho. \quad (3)$$

Since  $cost(EOPT, t)$  is a lower bound on the optimal cost, this condition ensures that the competitive ratio is at most  $\rho$  when  $RR(\rho)$  makes a rejection. This observation quickly leads to the following lemma:

**LEMMA 8**  *$RR(\rho)$  is  $\rho$ -competitive if it completes or rejects each job within time  $c$  of its release.*

**Proof:** Let  $OPT$  be an optimal psuedoschedule. We claim that each job is active in  $OPT$  at least as long as in  $RR$ . For jobs that  $OPT$  rejects, this follows immediately from our assumption on job completion/rejection times since these jobs are active in  $OPT$  for time  $c$  after their release. Suppose to the contrary that there exists a job that  $OPT$  runs before  $RR$ . Let  $J_i$  be the job with earliest completion time in  $RR$  having this property. Let  $J_j$  be the job  $RR$  completed immediately before  $J_i$ . Because  $RR$  runs the job with latest arrival time,  $r_j \geq r_i$ .  $OPT$  does not reject job  $J_j$  since it does not reject job  $J_i$ . Also, because it uses the same criteria to select which job to run,  $OPT$  must have completed job  $J_j$  before running job  $J_i$ . Thus,  $C_j^{OPT} < C_i^{OPT} \leq C_j^{RR}$ , contradicting our assumption that job  $J_i$  was the first job that  $OPT$  runs before  $RR$ .

Now the lemma follows by considering how the ratio  $cost(RR, t)/cost(EOPT, t)$  changes over time. Since each job is active in  $OPT$  at least as long as in  $RR$ ,  $cost(EOPT, t)$  increases at least as fast as  $cost(RR, t)$  except when  $RR$  rejects a job. Thus, the ratio declines toward 1 except when  $RR$  rejects a job. By definition, however,  $RR$  only rejects a job when doing so does not raise the ratio above  $\rho$ . Therefore, the ratio lies between 1 and  $\rho$  at all times. Since this ratio at the end of the input is exactly the competitive ratio, the competitive ratio of  $RR(\rho)$  is at most  $\rho$ .  $\square$

Using this, we can show that  $RR(2)$  is 2-competitive:

**THEOREM 9**  *$RR(2)$  is 2-competitive for unit-length jobs all having rejection cost  $c$ .*

**Proof:** Call all jobs that  $RR(\rho)$  does not run within time  $c$  of their release *extra* jobs. (Note that this includes all jobs  $RR$  rejects.) We show that all extra jobs are rejected within time  $c$  of their release; the result then follows from Lemma 8. First we consider the simple case when all extra jobs arrive at a single time. Without loss of generality, assume this time is 0. Let  $k$  be the number of extra jobs and let  $OPT$  be a normalized optimal solution.

Let  $t_i$  be the time of  $RR$ 's  $i^{\text{th}}$  rejection. For notational convenience, we define  $t_0 = 0$ . We show that the time between  $t_{i-1}$  and  $t_i$  is at most  $c/(k+i)$ . The result then follows from Lemma 1 since the  $i^{\text{th}}$  rejection occurs before  $\tau_{k,i}$  as defined in Equation 2. Consider how the ratio  $cost(RR, t)/cost(EOPT, t)$  changes between times  $t_{i-1}$  and  $t_i$ . All  $k$  extra jobs contribute to the denominator since they remain active in the pseudoschedule even if  $OPT$  rejects them. Only  $k - (i-1) = k+1-i$  of them contribute to the numerator since  $i-1$  have already been rejected. In addition to the extra jobs, there is at least one other job active at all times or  $RR$  would work on extra jobs. This job and any other non-extra job contributes equally to the numerator and denominator since  $OPT$  and  $RR$  process jobs in the same order. Thus, the rejection occurs latest if  $k+2-i$  jobs contribute to the numerator and  $k+1$  contribute to the denominator. Therefore, if  $RR$  has not made the  $i^{\text{th}}$  rejection before time  $t_{i-1} + c/(k+i)$ , the largest possible value of the ratio in Equation 3 is

$$\frac{cost(RR, t_{i-1}) + (k+2-i)c/(k+i) + c}{cost(EOPT, t_{i-1}) + (k+1)c/(k+i)} \leq 2$$

and  $RR$  makes the  $i^{\text{th}}$  rejection at that time.

It remains to show that each extra job is rejected within time  $c$  of its release when extra jobs arrive at several times. Suppose to the contrary that an input instance exists in which some job  $J$  is not completed or rejected within time  $c$  of its release. Since extra jobs arriving after job  $J$  only hasten its rejection, we assume the extra jobs arriving with  $J$  are the last extra jobs and the only ones not rejected within time  $c$  of their arrival. We also assume that non-extra jobs arrive one per unit time since deviations from this also hasten rejections. Call each set of extra jobs arriving concurrently a *group*. We give a modification of the instance that reduces the number of groups in such a way that every rejection after the arrival of the second group occurs no earlier. This suffices since repeatedly applying this transformation to our initial instance creates an instance with a single group of extra jobs that are not all rejected within time  $c$ , which we have already shown cannot occur.

Suppose the first two groups of extra jobs arrive at times 0 and  $t$ , respectively. Let  $k$  be the size of the first group and let  $i$  be the number of these jobs rejected by time  $t$ . Our transformation removes all jobs arriving before time  $t$  and adds  $k - i$  jobs to the second group. We show the rejections occur later in this modified instance by showing that  $\text{cost}(RR, t)$  grows at least as quickly and  $\text{cost}(EOPT, t)$  grows more slowly. The value of  $\text{cost}(RR, t)$  grows at least as quickly at time  $t$  since exactly the same number of jobs are active in  $RR$ . The faster growth continues after time  $t$  because slower growth of  $\text{cost}(EOPT, t)$  delays  $RR$ 's rejections. To show that  $\text{cost}(EOPT, t)$  grows more slowly, consider how the first group of extra jobs contribute to it after time  $t$  in the original instance. We showed above that an extra group with  $k$  jobs causes  $i$  rejections by time  $\sum_{j=1}^i c/(k+j) \leq ci/k$  after its arrival. Thus,  $t \leq ci/k$ . Since the first group has  $k$  jobs contributing to  $\text{cost}(EOPT, t)$  until time  $c$ , their total contribution after time  $t$  is at least  $ck(1 - i/k) = c(k - i)$ . This is exactly the total contribution of the  $k - i$  jobs we replaced them with. In addition, the replacement jobs contribute until time  $t + c$  rather than stopping at time  $c$ . Since the contribution of the replacement jobs is no larger and it is more dispersed,  $\text{cost}(EOPT, t)$  grows more slowly.  $\square$

## 4 NP-completeness of minimizing flow with rejections

Turning to the offline problem, we show that a restricted version of the problem is NP-complete.

**THEOREM 10** *Minimizing total flow time with rejections is NP-complete even when the cost of rejecting a job is proportional to its processing time.*

Note that this theorem implies that minimizing the sum of completion times with rejection is also NP-complete since the same schedule is optimal for total flow time and sum of completion times. In fact, our proof uses ideas of Du et al. [5], who proved that it is NP-complete to find a multiprocessor preemptive schedule minimizing the sum of completion times. Rejected jobs in our schedule are equivalent to jobs running on a second processor in theirs.

**Proof:** The problem is in NP because once the jobs to reject are specified, the others are run using SRPT [2]. To show hardness, we give a reduction from PARTITION [7]:

PARTITION: Given a set  $A = \{a_1, a_2, \dots, a_n\}$  of positive integers, does there exist a partition of  $A$  into  $A_1$  and  $A_2$  such that  $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$ ?

Let  $B = \sum_{i=1}^n a_i$ . We assume  $B$  is a multiple of 2 since otherwise a partition cannot exist. We also assume  $n \geq 5$  since small instances of PARTITION can be quickly decided. We create a scheduling instance with  $n + 2$  jobs. For each element  $a_i \in A$ , we create job  $J_i$  with  $r_i = 2(i - 1)B$ ,  $p_i = 2a_i$ , and  $c_i = 5a_i$ . In addition, there are two jobs  $J_{n+1}$  and  $J_{n+2}$  with  $r_{n+1} = 0$ ,  $p_{n+1} = 2nB$ ,  $c_{n+1} = 5nB$ ,  $r_{n+2} = 2nB + B$ ,  $p_{n+2} = 2nB$ , and  $c_{n+2} = 5nB$ . Observe that the rejection cost for each job is  $5/2$  times its processing time.

We claim that a partition exists if and only if a schedule exists with cost at most  $(4n + 9/2)B$ . Suppose a partition of  $A$  into  $A_1$  and  $A_2$  exists. Let  $S_1$  and  $S_2$  be sets of jobs containing the jobs corresponding to  $A_1$  and  $A_2$  respectively. We create a schedule that rejects the jobs in  $S_1$  as they arrive and runs the jobs in  $S_2$  to completion as they arrive. In the times between these executions, it runs  $J_{n+1}$ . Since the jobs in  $S_2$  have total execution time  $\sum_{J_i \in S_2} p_i = \sum_{a_i \in A_2} 2a_i = B$ , job  $J_{n+1}$  finishes at time  $2nB + B$ . Thus, we can run job  $J_{n+2}$  as soon as it arrives. This schedule has total cost  $\sum_{J_i \in S_2} p_i + \sum_{J_i \in S_1} c_i + (2nB + B) + 2nB = B + (5/2)B + 4nB + B = (4n + 9/2)B$ .

Now suppose a schedule with cost at most  $(4n + 9/2)B$  exists. Clearly, neither  $J_{n+1}$  nor  $J_{n+2}$  are rejected. Let  $S_1$  be the set of rejected jobs and  $S_2$  be the other members of  $\{J_1, J_2, \dots, J_n\}$ . Also, let  $A_1$  and  $A_2$  contain the  $a_i$  corresponding to members of  $S_1$  and  $S_2$  respectively. We assume that each member  $J_i$  of  $S_2$  is run immediately following its arrival since otherwise doing so and running  $J_{n+1}$  when  $J_i$  was run produces a schedule with no greater cost. Similarly, we assume that job  $J_{n+1}$  is completed before  $J_{n+2}$  is started. If  $\sum_{a_i \in A_2} a_i > B/2$ , then  $J_{n+1}$  does not complete before the arrival of  $J_{n+2}$  and so the cost is at least  $5 \sum_{a_i \in A_1} a_i + 2 \sum_{a_i \in A_2} a_i + (2nB + 2 \sum_{a_i \in A_2} a_i) + (2nB + 2 \sum_{a_i \in A_2} a_i - B) = 4nB + 4B + \sum_{a_i \in A_2} a_i > (4n + 9/2)B$ . (Recall that  $\sum_{a_i \in A_1} a_i + \sum_{a_i \in A_2} a_i = B$ .) However, if  $\sum_{a_i \in A_2} a_i < B/2$ , then the cost is at least  $5 \sum_{a_i \in A_1} a_i + 2 \sum_{a_i \in A_2} a_i + (2nB + 2 \sum_{a_i \in A_2} a_i) + 2nB = 4nB + 5B - \sum_{a_i \in A_2} a_i > (4n + 9/2)B$ . Thus,  $\sum_{a_i \in A_2} a_i = B/2 = \sum_{a_i \in A_1} a_i$  so  $A_1$  and  $A_2$  form a partition of  $A$ .  $\square$

## 5 Discussion

Although we have made progress on several parts of this problem, there remain areas for further work. We are most interested in closing the gap between the lower bound of  $e/(e-1) \approx 1.582$  and the various 2-competitive algorithms for unit-length jobs with rejection cost  $c$ . We believe that  $RR$  can be better than 2-competitive; certainly there is slack for a single group of extra jobs, which  $RR(2)$  finishes by approximately  $0.7c$ . There is also still a gap between our lower bound of  $(2 + \sqrt{2})/2 \approx 1.707$  and the 2-competitive algorithm of Bansal et al. [2] for jobs with arbitrary processing times and rejection cost  $c$ . In addition, it would be interesting to know if randomized algorithms can beat the competitive ratios given in Theorems 3 and 4. In the offline setting, we have shown that one form of the problem is NP-complete, but it is not known whether the problem remains NP-complete when all jobs have rejection cost  $c$ . Another open question is whether either the full problem or this special case admit a PTAS, i.e. a  $(1 + \epsilon)$ -approximation algorithm that runs in polynomial-time for any constant  $\epsilon$ .

More generally, much more can be done to address the question of how to combine planning with scheduling. This question has not been considered in many scheduling settings nor has anyone characterized the requirements of a specific application.

## Acknowledgments

The author thanks Dan Cranston for help solving the recurrence giving the lower bound  $e/(e-1)$ . He also thanks Cindy Phillips, Vitus Leung, Kevin Milans, John Fischer, and the anonymous referees for helpful comments about revising the paper.

## References

- [1] N. Bansal. *Algorithms for flow time scheduling*. PhD thesis, Carnegie Mellon University, 2003. <http://www.research.ibm.com/people/n/nikhil/papers/thesis.pdf>.
- [2] N. Bansal, A. Blum, S. Chawla, and K. Dhamdhere. Scheduling for flow-time with admission control. In *Proc. 11th Annual European Symp. Algorithms*, number 2832 in LNCS, pages 43–54, 2003.
- [3] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. In *Proc. 14th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 508–516, 2003.

- [4] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. In *Proc. 7th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 95–103, 1996.
- [5] J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraints. *Theoretical Computer Science*, 75(3):347–355, 1990.
- [6] D. Engels, D. Karger, S. Kolliopoulos, S. Sengupta, R. Uma, and J. Wein. Techniques for scheduling with rejection. In *Proc. 6th Annual European Symp. Algorithms*, volume 1461 of *LNCS*, pages 490–501, 1998.
- [7] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.
- [8] H. Hoogeveen, M. Skutella, and G. Woeginger. Preemptive scheduling with rejection. In *Proc. 8th Annual European Symp. Algorithms*, volume 1879 of *LNCS*, pages 268–277, 2000.
- [9] I.M. Ovacik and R. Uzsoy. *Decomposition methods for complex factory scheduling problems*. Kluwer Academic Publishers, 1997.
- [10] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J.Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 15, pages 15–1–15–41. CRC Press, 2004.
- [11] W. Rudin. *Principles of mathematical analysis*. McGraw-Hill, Inc., 3<sup>rd</sup> edition, 1976.
- [12] L. Schrage. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research*, 16:687–690, 1968.
- [13] S.S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 262:437–458, 2001.
- [14] S. Sengupta. Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection. In *Proc. 9th Workshop on Algorithms and Data Structures*, number 2748 in *LNCS*, pages 79–90, 2003.
- [15] D. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197–199, 1976.

## Appendix: Proof of Lemmas 1 and 2

Before proving our main results, we show the following technical lemma:

**LEMMA 11** *Consider the axis-aligned rectangle with corners  $(i, 1/i)$  and  $(i + 1, 1/(i + 1))$ . The proportion of its area above the function  $f(x) = 1/x$  is at most  $1/2 + 1/(2i)$ .*

**Proof:** The area of the box is  $1/i - 1/(i + 1) = 1/(i(i + 1))$ . The area within the box and under the curve is

$$\int_i^{i+1} \frac{dx}{x} - \frac{1}{i+1} = \ln \frac{i+1}{i} - \frac{1}{i+1}$$

Thus, the proportion of area under the curve is  $1 - i(i+1) \ln((i+1)/i) + i = 1 + i - i(i+1) \ln(1 + 1/i)$ . Using the first two terms of the Taylor series expansion for  $\ln(1 + x)$ , this is at most

$$1 + i - (i^2 + i) \left( \frac{1}{i} - \frac{1}{2i^2} \right) = 1/2 + 1/(2i)$$

□

This allows us to prove the first of our lemmas about  $\tau_{k,k}$ .

**Proof of Lemma 1:** For the value of  $\tau_{k,k}$ , we use Equation 2:

$$\tau_{k,k} = \sum_{j=1}^k \frac{1}{(k+1)/(e-1) + j - 1} = \int_1^{k+1} \frac{dx}{(k+1)/(e-1) + \lfloor x \rfloor - 1}$$

We want to remove the floor from this integral, but doing so decreases the value. We bound the difference by considering the total area in the axis-aligned rectangles with corners along the curve

$y = 1/x$  at  $x = (k+1)/(e-1) + i$  for  $i = 0, 1, \dots, k$ . These boxes have total area

$$\sum_{i=(k+1)/(e-1)}^{(k+1)/(e-1)+k} \left( \frac{1}{i} - \frac{1}{i+1} \right) = \frac{e-1}{k+1} - \frac{e-1}{ke+1}$$

When  $k \geq 10$ , the proportion of this area that is part of the error is at most

$$\frac{1}{2} + \frac{1}{2i} \leq \frac{1}{2} + \frac{e-1}{22} = \frac{e+10}{22} \leq \frac{1}{e-1}$$

by Lemma 11. Thus,  $\tau_{k,k}$  for  $k \geq 10$  obeys

$$\begin{aligned} \tau_{k,k} &\leq \int_1^{k+1} \frac{dx}{(k+1)/(e-1) + x - 1} + \frac{1}{e-1} \left( \frac{e-1}{k+1} - \frac{e-1}{ke+1} \right) \\ &= \int_1^{k+1} \frac{(e-1)dx}{k+2-e+x(e-1)} + \frac{k(e-1)}{(k+1)(ke+1)} \\ &< \ln(k+2-e+(k+1)(e-1)) - \ln(k+2-e+(e-1)) + \frac{e-1}{ke+1} \\ &= \ln \frac{ke+1}{k+1} + \frac{e-1}{ke+1} = 1 - \ln \frac{ke+e}{ke+1} + \frac{e-1}{ke+1} \\ &= 1 - \ln \left( 1 + \frac{e-1}{ke+1} \right) + \frac{e-1}{ke+1} \end{aligned}$$

The last line is at most 1 since  $\ln(1+x) = \sum_{i=1}^{\infty} (-1)^{i+1} x^i / i$ , which is greater than  $x$  for  $x < 1$ .

For  $k < 10$ , we verify the identity by hand using Equation 2:  $\tau_{1,1} \approx 0.859$ ,  $\tau_{2,2} \approx 0.937$ ,  $\tau_{3,3} \approx 0.961$ ,  $\tau_{4,4} \approx 0.972$ ,  $\tau_{5,5} \approx 0.979$ ,  $\tau_{6,6} \approx 0.983$ ,  $\tau_{7,7} \approx 0.985$ ,  $\tau_{8,8} \approx 0.987$ ,  $\tau_{9,9} \approx 0.989$ .  $\square$

The other observation follows quickly from this:

### Proof of Lemma 2:

$$\begin{aligned} \tau_{k,k} &= \sum_{j=1}^k \frac{1}{(k+1)/(e-1) + j - 1} \\ &\geq \sum_{j=1}^k \frac{1}{\lceil (k+1)/(e-1) \rceil + j - 1} \\ &= H_{\lceil (k+1)/(e-1) \rceil + k - 1} - H_{\lceil (k+1)/(e-1) \rceil - 1} \end{aligned}$$

where  $H_n = \sum_{j=1}^n 1/j$  is the  $n^{\text{th}}$  Harmonic number. Recall that  $\lim_{n \rightarrow \infty} (H_n - \ln n) = \gamma$ , where  $\gamma \approx 0.577$  is Euler's constant [11]. Thus,

$$\begin{aligned} \lim_{k \rightarrow \infty} \tau_{k,k} &\geq \lim_{k \rightarrow \infty} (H_{\lceil (k+1)/(e-1) \rceil + k - 1} - H_{\lceil (k+1)/(e-1) \rceil - 1}) \\ &= \lim_{k \rightarrow \infty} (\ln((k+1)/(e-1) + k - 1) - \ln((k+1)/(e-1) - 1)) \\ &= \lim_{k \rightarrow \infty} \ln \frac{(k+1)/(e-1) + k - 1}{(k+1)/(e-1) - 1} = \ln(1 + e - 1) \\ &= 1 \end{aligned}$$

The result then follows from Lemma 1.  $\square$

# ONLINE SCHEDULING OF PRECEDENCE CONSTRAINED TASKS

Yumei Huo and Joseph Y-T. Leung

Department of Computer Science  
New Jersey Institute of Technology  
Newark, NJ 07102, USA

**Abstract** A fundamental problem in scheduling theory is that of scheduling a set of  $n$  tasks, with precedence constraints, on  $m \geq 1$  identical and parallel processors so as to minimize the makespan (schedule length). In the past, research has focussed on the setting where all tasks are available for processing at the beginning (i.e., at time  $t = 0$ ). In this article we consider the situation where tasks, along with their precedence constraints, are released at different times, and the scheduler has to make scheduling decisions without knowledge of future releases. In other words, the scheduler has to schedule tasks in an online fashion. We consider both preemptive and nonpreemptive schedules. We show that *optimal* online algorithms exist for some cases, while for others it is impossible to have one. Our results give a sharp boundary delineating the possible and the impossible cases.

**Keywords:** schedule length, preemptive and nonpreemptive scheduling, online and offline scheduling, parallel and identical processors, precedence constraints, intrees and outtrees.

## 1. Introduction

A fundamental problem in scheduling theory is that of scheduling a set of  $n$  tasks, with precedence constraints, on  $m \geq 1$  identical and parallel processors so as to minimize the makespan (schedule length). In this setting we are given  $n$  tasks,  $1, 2, \dots, n$ , where each task  $j$  has a processing time  $p_j$ . The tasks have precedence constraints,  $\prec$ , in that  $i \prec j$  signifies that task  $j$  cannot start until task  $i$  is finished. The tasks, together with their precedence constraints, are described by a directed acyclic graph  $G = (V, A)$ , where  $V$  is a set of vertices representing the tasks and  $A$  is a set of directed arcs representing the precedence constraints; there is a directed arc from task  $i$  to task  $j$  if  $i \prec j$ . We assume the graph has no transitive edges. The tasks can be scheduled preemptively or nonpreemptively. In preemptive scheduling, a task can be interrupted before it completes and later resumed on a possibly different processor. We assume that there is no time loss in preemption. By contrast, in nonpreemptive scheduling, a task once begun execution can not be interrupted until it completes. With respect to a schedule  $S$ , the completion time of task  $i$  is denoted by  $C_i$ , and the makespan is denoted by  $C_{\max} = \max\{C_i\}$ . The goal is to schedule the set of tasks on  $m \geq 1$  identical and parallel processors so as to minimize  $C_{\max}$ . In the three-field classification scheme introduced by Graham *et al.* [4], the problems considered in this article are  $P \mid prec \mid C_{\max}$  and  $P \mid pmtn, prec \mid C_{\max}$ .

A task  $i$  is said to be an *immediate predecessor* of another task  $j$  if there is a directed arc  $(i, j)$  in  $G$ ;  $j$  is said to be an *immediate successor* of  $i$ . Task  $i$  is said to be a *predecessor* of task  $j$  if there is a directed *path* from  $i$  to  $j$  in  $G$ ;  $j$  is said to be a *successor* of  $i$ . We say that  $G$  is an *intree* if each vertex, except the root, has exactly one immediate successor.  $G$  is an *outtree* if each vertex, except the root, has exactly one immediate predecessor. A *chain* is an outtree where each vertex has at most one immediate successor. We use *prec* to denote an arbitrary directed acyclic graph.

In the past, research in scheduling theory has concentrated on these four classes of precedence constraints: *prec*, *intree*, *outtree*, and *chains*. A number of polynomial-time optimal algorithms have been developed. In nonpreemptive scheduling, the famous Coffman-Graham algorithm [1] is optimal for  $P2 \mid p_j = 1, \textit{prec} \mid C_{\max}$ , while the well-known Hu's algorithm [6] is optimal for  $P \mid p_j = 1, \textit{intree} \mid C_{\max}$  and  $P \mid p_j = 1, \textit{outtree} \mid C_{\max}$ ; we will describe these two algorithms in detail in the next section. It is known that  $P \mid p_j = 1, \textit{prec} \mid C_{\max}$  is strongly NP-hard [3], although the complexity is still open for each fixed  $m \geq 3$ . If the tasks have arbitrary processing times, then the problem becomes NP-hard in the ordinary sense even if we have two processors and the tasks are independent; i.e.  $P2 \parallel C_{\max}$  is NP-hard in the ordinary sense [3].

For preemptive scheduling, the Muntz-Coffman algorithm [8, 9] is optimal for  $P2 \mid \textit{pmtn}, \textit{prec} \mid C_{\max}$ ,  $P \mid \textit{pmtn}, \textit{intree} \mid C_{\max}$  and  $P \mid \textit{pmtn}, \textit{outtree} \mid C_{\max}$ ; we will describe this algorithm in detail in Section 1.3. Again,  $P \mid \textit{pmtn}, \textit{prec} \mid C_{\max}$  is strongly NP-hard [3], while the complexity is still open for each fixed  $m \geq 3$ .

All of the algorithms mentioned above assume that all tasks are available for processing at the beginning (i.e., at time  $t = 0$ ). In this article, we consider the situation where tasks, along with their precedence constraints, are released at different times, and the scheduler has to make scheduling decision without knowledge of future releases. In other words, the scheduler has to schedule tasks in an online fashion. We say that an online scheduling algorithm is *optimal* if it always produces a schedule with the minimum  $C_{\max}$ , i.e., a schedule as good as any schedule produced by any scheduling algorithm with full knowledge of future releases of tasks. Since an online scheduling algorithm has to schedule tasks in an online fashion, it is not clear that an optimal online scheduling algorithm necessarily exists. In this article, we show that online scheduling algorithms exist for some cases, while for others it is impossible to have one. Our results give a sharp boundary delineating the possible and the impossible cases.

We extend the notation of Graham *et al.* [4] to online scheduling problems in a natural way. For example,  $P2 \mid p_j = 1, \textit{prec}_i \text{ released at } r_i \mid C_{\max}$  refers to the case where tasks with arbitrary precedence constraint,  $\textit{prec}_i$ , are released at time  $r_i$ . In this case, there are two processors, each task has unit processing time, and preemption is not allowed. As another example,  $P \mid \textit{pmtn}, \textit{outtree}_i \text{ released at } r_i \mid C_{\max}$  refers to the case where tasks with outtree precedence constraint,  $\textit{outtree}_i$ , are released at time  $r_i$ . In this case, there are arbitrary number of processors, tasks have arbitrary processing times, and preemption is allowed.

Hong and Leung [5] have given an optimal online scheduling algorithm for a set of independent tasks on an arbitrary number of processors where preemption is allowed. The idea of their algorithm is to schedule tasks using a modified McNaughton's wrap-around rule. (It is known that McNaughton's wrap-around rule is optimal for  $P \mid \textit{pmtn} \mid C_{\max}$  [7].) Tasks will be executed according to the schedule until new tasks arrive, at which time the algorithm will reschedule, by the same rule, the remaining portions of the unfinished tasks along with the newly arrived tasks. This process is repeated until all tasks are finished and no new tasks arrive.

Note that for nonpreemptive scheduling, it can be shown that it is impossible to have an optimal online algorithm for a set of independent tasks with arbitrary processing times, even if there are only two processors.

In this article, we show that optimal online scheduling algorithms exist for:

- (1)  $P2 \mid p_j = 1, \textit{prec}_i \text{ released at } r_i \mid C_{\max}$ .
- (2)  $P \mid p_j = 1, \textit{outtree}_i \text{ released at } r_i \mid C_{\max}$ .
- (3)  $P2 \mid \textit{pmtn}, \textit{prec}_i \text{ released at } r_i \mid C_{\max}$ .
- (4)  $P \mid \textit{pmtn}, \textit{outtree}_i \text{ released at } r_i \mid C_{\max}$ .

Using an adversary argument, we show that it is impossible to have optimal online scheduling algorithms for:

- (1)  $P3 \mid p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$ .
- (2)  $P2 \mid p_j = p, \text{chains}_i \text{ released at } r_i \mid C_{\max}$ .
- (3)  $P3 \mid \text{pmtn}, p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$ .

All of our optimal online scheduling algorithms follow the same format as the algorithm given in Hong and Leung [5]. For  $P2 \mid p_j = 1, \text{prec}_i \text{ released at } r_i \mid C_{\max}$ , we use the Coffman-Graham algorithm to schedule tasks, and follow the schedule to execute tasks until new tasks arrive, at which time we reschedule the remaining portions of the unfinished tasks along with the newly arrived tasks. We use Hu's algorithm for  $P \mid p_j = 1, \text{outtree}_i \text{ released at } r_i \mid C_{\max}$ , and the Muntz-Coffman algorithm for  $P2 \mid \text{pmtn}, \text{prec}_i \text{ released at } r_i \mid C_{\max}$  and  $P \mid \text{pmtn}, \text{outtree}_i \text{ released at } r_i \mid C_{\max}$ .

The organization of this article is as follows. In the next section we consider nonpreemptive scheduling, while preemptive scheduling will be considered in Section 3. Finally, we draw some conclusions in the last section.

## 2. Nonpreemptive schedules

In this section we consider nonpreemptive scheduling only. We first show that it is impossible to have optimal online algorithms for  $P3 \mid p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$  and  $P2 \mid p_j = p, \text{chains}_i \text{ released at } r_i \mid C_{\max}$ . We then give an optimal online algorithm for  $P2 \mid p_j = 1, \text{prec}_i \text{ released at } r_i \mid C_{\max}$  in Section 1.2.1 and an optimal online algorithm for  $P \mid p_j = 1, \text{outtree}_i \text{ released at } r_i \mid C_{\max}$  in Section 1.2.2.

**THEOREM 1** *It is impossible to have an optimal online algorithm for  $P3 \mid p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$ .*

**Proof :** We will use adversary argument to prove the theorem. Consider the intrees shown in Fig. 1: The number of processors is three,  $\text{intree}_1$  is released at  $r_1 = 0$  and  $\text{intree}_2$  is released at  $r_2 = 4$ .

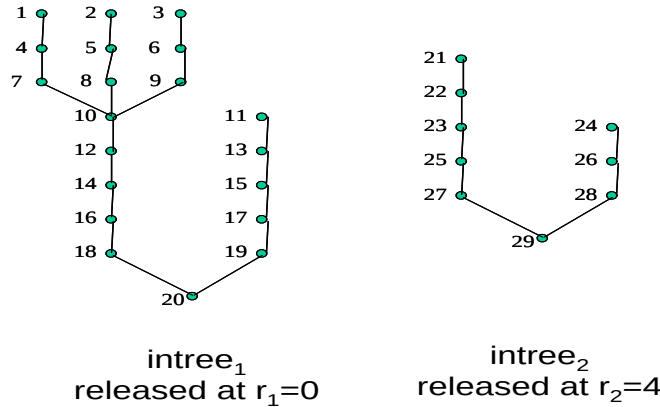


Figure 1. Example showing impossibility for  $P3 \mid p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$

For  $\text{intree}_1$ , the length of the longest path is nine, so the makespan can not be smaller than nine. To obtain the minimum makespan, task 10 must finish by time  $t = 4$ , which means that all its predecessors must be finished by time  $t = 3$ . Since task 10 has nine predecessors and since there are only three processors, all of the predecessors of task 10 must be executed in the first three time units. This means that there must be an idle processor in the time interval  $[3, 4]$ . Now, if  $\text{intree}_2$  is



released at time  $t = 4$ , then the makespan must be larger than 10. As shown in Fig. 2, the optimal makespan is 10. On the other hand, if task 10 is not completed by time  $t = 4$ , then the schedule is already not optimal for  $intree_1$ .

|    |   |   |   |    |    |    |    |    |    |    |    |    |    |
|----|---|---|---|----|----|----|----|----|----|----|----|----|----|
|    | 0 | 1 | 2 | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 |
| S1 | 1 | 4 | 7 | 10 | 12 | 14 | 16 | 17 | 18 | 20 | 29 |    |    |
|    | 2 | 5 | 8 | 11 | 13 | 15 | 24 | 25 | 19 | 28 |    |    |    |
|    | 3 | 6 | 9 |    | 21 | 22 | 23 | 26 | 27 |    |    |    |    |

|    |    |    |    |   |    |    |    |    |    |    |    |    |    |
|----|----|----|----|---|----|----|----|----|----|----|----|----|----|
|    | 0  | 1  | 2  | 3 | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 |
| S2 | 1  | 3  | 5  | 7 | 10 | 12 | 14 | 16 | 18 | 20 |    |    |    |
|    | 2  | 4  | 6  | 8 | 21 | 17 | 23 | 25 | 19 | 29 |    |    |    |
|    | 11 | 13 | 15 | 9 | 24 | 22 | 26 | 28 | 27 |    |    |    |    |

Figure 2. Schedule for the example in Fig. 1

Thus, the adversary first releases  $intree_1$  at time  $t = 0$ . If the online algorithm did not finish task 10 by time  $t = 4$ , then the schedule produced by the online algorithm is already not optimal for  $intree_1$ . On the other hand, if the online algorithm completes task 10 by time  $t = 4$ , then the adversary releases  $intree_2$  at time  $t = 4$ . The online algorithm cannot finish both intrees by time  $t = 10$ , but the optimal makespan is 10. Again, the online algorithm did not produce an optimal schedule.  $\square$

**THEOREM 2** *It is impossible to have an optimal online algorithm for  $P2 \mid p_j = p, chains_i$  released at  $r_i \mid C_{\max}$ .*

**Proof :** Consider the chains shown in Fig. 3: Two chains released at  $r_1 = 0$ , one chain released at  $r_2 = 5$ , each task in the chains has two units of processing time, and the number of processors is two. The minimum makespan for the first two chains (released at time  $t = 0$ ) is six, which can be attained only if both chains execute continuously from time  $t = 0$  until time  $t = 6$ . Now, if the second chain is released at time  $t = 5$ , then the makespan will be 16. However, as shown in Fig. 3, the optimal makespan is 15.

Thus, the adversary first releases the two chains at time  $t = 0$ . If the online algorithm leaves a processor idle in the time interval  $[0, 5]$ , then the schedule is already not optimal for the two chains. On the other hand, if the online algorithm keeps both processors busy during the interval  $[0, 5]$ , then the adversary releases the second chain at time  $t = 5$ . The online algorithm cannot finish all the chains by time 15, but the optimal makespan is 15. Again, the online algorithm did not produce an optimal schedule.  $\square$

## 2.1 UET tasks, arbitrary precedence constraint and two processors

The Coffman-Graham algorithm is optimal for  $P2 \mid p_j = 1, prec \mid C_{\max}$ . It works by first assigning a label to each task which corresponds to the priority of the task; tasks with higher labels have higher priority. Once the labels are assigned, tasks are scheduled as follows: Whenever a processor becomes free, assign that task all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned.

Before we describe the labeling algorithm, we need to define a linear order on decreasing sequences of positive integers as follows.

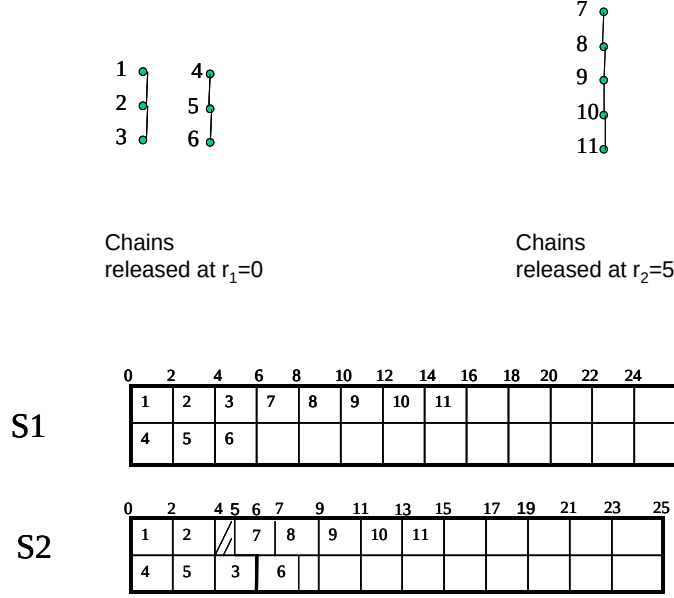


Figure 3. Example showing impossibility for  $P2 \mid p_j = p, \text{chains}_i \text{ released at } r_i \mid C_{\max}$

**DEFINITION 3** Let  $N = (n_1, n_2, \dots, n_t)$  and  $N' = (n'_1, n'_2, \dots, n'_{t'})$  be two decreasing sequences of positive integers. We say that  $N < N'$  if either

1. For some  $i$ ,  $1 \leq i \leq t$ , we have  $n_j = n'_j$  for all  $j$  satisfying  $1 \leq j \leq i - 1$  and  $n_i < n'_i$ , or
2.  $t < t'$  and  $n_j = n'_j$  for all  $j$  satisfying  $1 \leq j \leq t$ .

*Example:*  $(8, 6, 4, 3) < (8, 6, 5)$  and  $(9, 8, 6) < (9, 8, 6, 4, 3)$ .

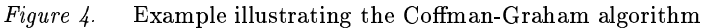
Let  $n$  denote the number of tasks in  $prec$ . The labeling algorithm assigns to each task  $i$  an integer label  $\alpha(i) \in \{1, 2, \dots, n\}$ . The mapping  $\alpha$  is defined as follows. Let  $IS(i)$  denote the set of immediate successors of task  $i$  and let  $N(i)$  denote the decreasing sequence of integers formed by ordering the set  $\{\alpha(j) \mid j \in IS(i)\}$ .

1. An arbitrary task  $i$  with  $IS(i) = \emptyset$  is chosen and  $\alpha(i)$  is defined to be 1.
2. Suppose for some  $k \leq n$  that the integers  $1, 2, \dots, k - 1$  have been assigned. From the set of tasks for which  $\alpha$  has been defined on all elements of their immediate successors, choose the task  $j$  such that  $N(j) \leq N(i)$  for all such tasks  $i$ . Define  $\alpha(j)$  to be  $k$ .
3. Repeat the assignment in 2 until all tasks of  $prec$  have been assigned some integer.

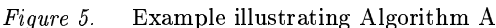
*Example:* Fig. 4 shows a set of tasks with their precedence constraints. The number inside the circle represents the task's index and the number next to the circle represents the label assigned to the task by the Coffman-Graham labeling algorithm. The schedule on two processors is also shown in Fig. 4.

Our online algorithm utilizes the Coffman-Graham algorithm to schedule tasks. When new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by the Coffman-Graham algorithm again.

## Algorithm A



*Example:* Fig. 5 shows another set of tasks released at time  $r_2 = 2$ , after the tasks in Fig. 4 were released at time  $r_1 = 0$ . Note that tasks 4, 5, 7, 8, and 9 from the first release are unfinished at time  $t = 2$ . They are rescheduled, along with the new tasks from the second release, by the Coffman-Graham algorithm. The final schedule obtained by Algorithm A is also shown.



The next lemma, whose proof will be omitted, is instrumental in proving that Algorithm A is optimal.

LEMMA 4 *Let  $S$  be a schedule for a set of tasks with arbitrary precedence constraints, where each task has unit processing time.*

1. If  $S$  has the largest number of tasks completed at any time instant  $t$ , then  $S$  must be optimal for  $C_{\max}$ .
2. If  $S$  has the minimum idle processor time at any time instant  $t$ , then  $S$  must be optimal for  $C_{\max}$ .
3. For two processors, the schedule  $S'$  produced by the Coffman-Graham algorithm has the largest number of tasks completed at any time instant  $t$ .

**THEOREM 5** *Algorithm A is optimal for  $P2 \mid p_j = 1, prec_i \text{ released at } r_i \mid C_{\max}$ . Moreover, the schedule produced by Algorithm A has the largest number of tasks completed at any time instant  $t$ .*

## 2.2 UET tasks, outtrees and arbitrary number of processors

Hu's algorithm is optimal for  $P \mid p_j = 1, outtree \mid C_{\max}$ . Like the Coffman-Graham algorithm, it first assigns a label to each task which corresponds to the priority of the task; tasks with higher labels have higher priority. Once the labels are assigned, tasks are scheduled as follows: Whenever a processor becomes free, assign that task all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned. In Hu's algorithm, the label of a task is a function of the *level* of the task.

**DEFINITION 6** *The level of a task  $i$  with no immediate successor is its processing time  $p_i$ . The level of a task with immediate successor(s) is its processing time plus the maximum level of its immediate successor(s).*

Hu's labeling algorithm assigns higher labels to tasks at higher levels; ties are broken in an arbitrary manner.

*Example:* Fig. 6 shows a set of tasks with outtree precedence constraint. The number inside the circle is the task's index and the number next to the circle is the label given by Hu's algorithm. A schedule on 4 processors produced by Hu's algorithm is also shown.

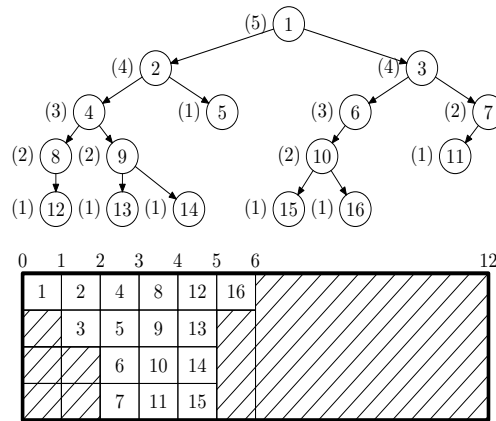


Figure 6. Example illustrating Hu's algorithm

Our online algorithm utilizes Hu's algorithm to schedule tasks. When new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by Hu's algorithm again.

## Algorithm B

Whenever new tasks arrive, do {  
 $t \leftarrow$  the current time;  
 $U \leftarrow$  the set of tasks active (i.e., not finished) at time  $t$ ;  
 Call Hu's algorithm to reschedule the tasks in  $U$ ;  
}

*Example:* Fig. 7 shows another outtree released at time  $r_2 = 3$ , after the outtree shown in Fig. 6 was released at time  $r_1 = 0$ . The schedule produced by Algorithm B is also shown.

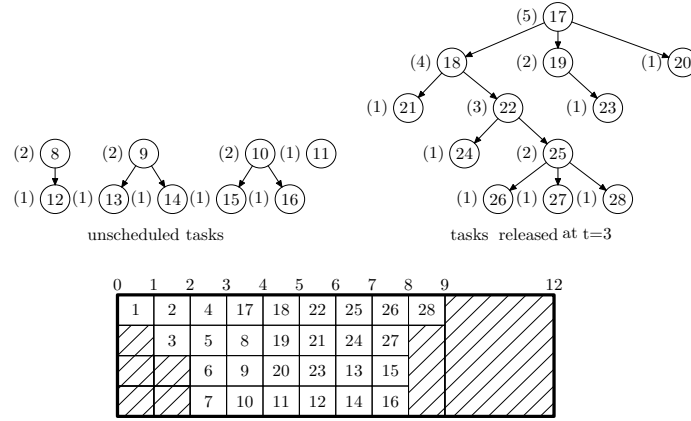


Figure 7. Example illustrating Algorithm B

**THEOREM 7** *Algorithm B is optimal for  $P \mid p_j = 1, \text{outtree}_i \text{ released at } r_i \mid C_{\max}$ . Moreover, the schedule produced by Algorithm B has the largest number of tasks completed at any time instant  $t$ .*

### 3. Preemptive schedules

In this section we consider preemptive scheduling only. We first show that it is impossible to have an optimal online algorithm for  $P3 \mid pmtn, p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$ . We then give an optimal online algorithm for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$  and  $P \mid pmtn, \text{outtree}_i \text{ released at } r_i \mid C_{\max}$ .

**THEOREM 8** *It is impossible to have an optimal online algorithm for  $P3 \mid pmtn, p_j = 1, \text{intree}_i \text{ released at } r_i \mid C_{\max}$ .*

This theorem can be shown by the same example and same argument as in Theorem 1 since preemption won't help in this example.

The Muntz-Coffman algorithm is optimal for the problems  $P2 \mid pmtn, prec \mid C_{\max}$ ,  $P \mid pmtn, \text{intree} \mid C_{\max}$ , and  $P \mid pmtn, \text{outtree} \mid C_{\max}$ . It is essentially a higher-level-first strategy; see Section 2.2 for the definition of level.

**Muntz-Coffman algorithm:** Assign one processor each to the tasks at the highest level. If there is a tie among  $y$  tasks (because they are at the same level) for the last  $x$  ( $x < y$ ) processors, then assign  $\frac{x}{y}$  processor to each of these  $y$  tasks. Whenever either of the two events below occurs,

reassign the processors to the unexecuted portion of the unfinished tasks according to the above rule. These are

1. A task is completed.
2. We reach a point where, if we were to continue the present assignment, we would be executing some tasks at a lower level at a faster rate than other tasks at a higher level.

The schedule produced by the Muntz-Coffman algorithm is a processor-sharing schedule. We can convert it to a preemptive schedule by marking the time instants where processor assignment change, and rescheduling the tasks executed between two adjacent time instants by McNaughton's wrap-around rule.

*Example:* Fig. 8 shows a set of tasks with their precedence constraints. Inside each circle is the name of the task and its processing time. The number next to each circle is the level of the task. The processor-sharing schedule on two processors produced by the Muntz-Coffman algorithm is shown. Finally, the preemptive schedule constructed from the processor-sharing schedule is also shown.

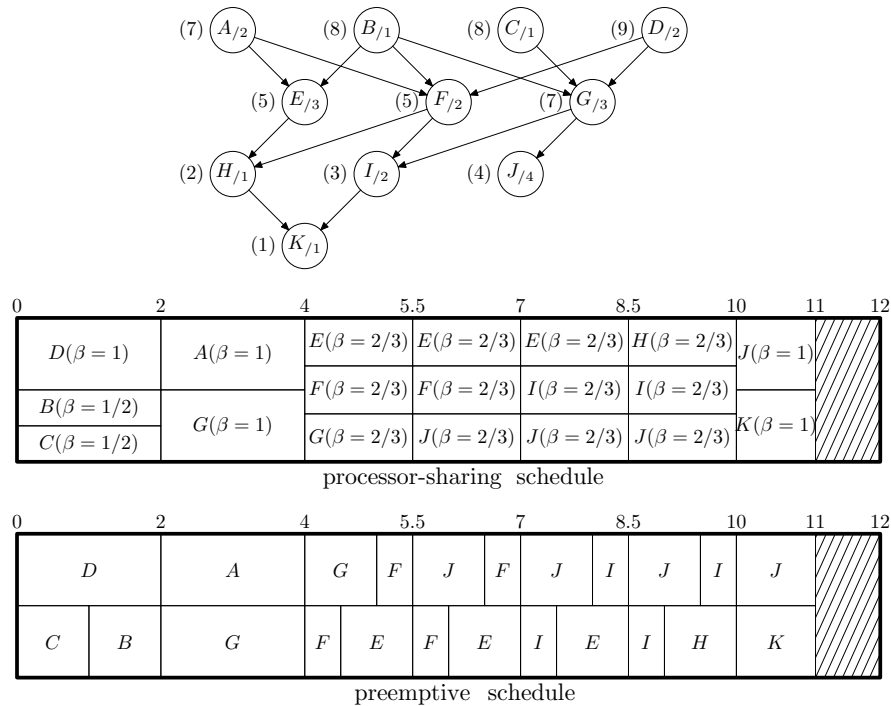


Figure 8. Example illustrating the Muntz-Coffman algorithm

Our online algorithm utilizes the Muntz-Coffman algorithm to schedule tasks. When new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by the Muntz-Coffman algorithm again.

### Algorithm C

Whenever new tasks arrive, do {

$t \leftarrow$  the current time;

$U \leftarrow$  the set of tasks active (i.e., not finished) at time  $t$ ;

Call the Muntz-Coffman algorithm to reschedule the tasks in  $U$ ;

}

Fig. 9 shows another set of tasks released at time  $r_2 = 6$ , after the tasks in Fig. 8 were released at time  $r_1 = 0$ . Algorithm C reschedules the unfinished portion of the unfinished tasks along with the new tasks. The processor-sharing schedule and the preemptive schedule are also shown.

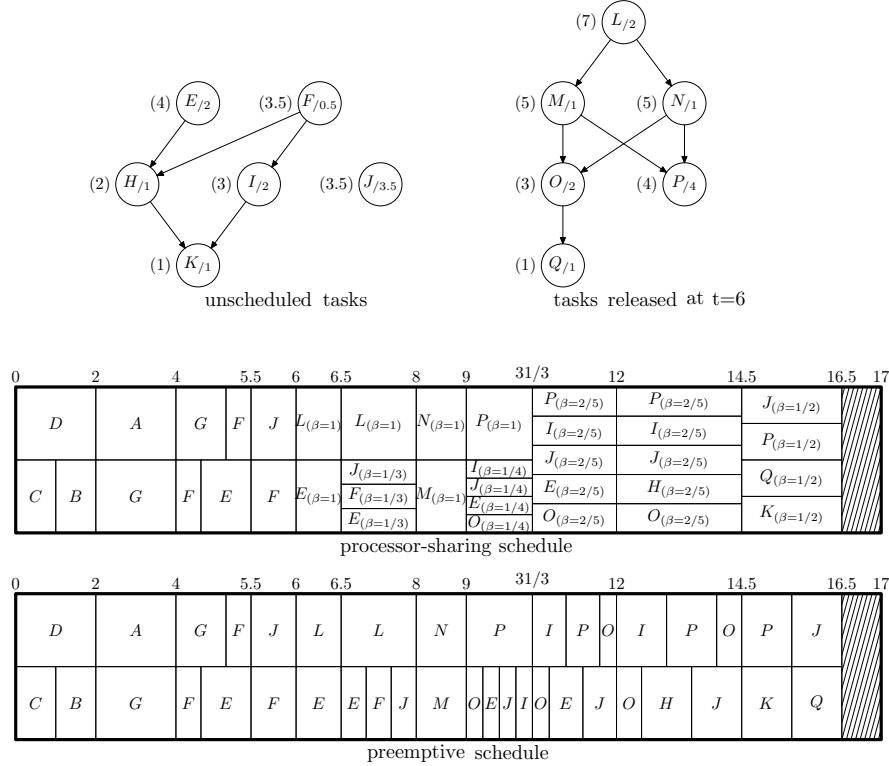


Figure 9. Example illustrating Algorithm C

Before we give the theorems that Algorithm C is optimal for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$  and  $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$ , we will give an optimal offline algorithm for these two cases.

**Algorithm D:** Assign one processor each to the tasks at the highest level. If there is a tie among  $y$  tasks (because they are at the same level) for the last  $x$  ( $x < y$ ) processors, then assign  $\frac{x}{y}$  processor to each of these  $y$  tasks. Whenever one of the three events below occurs, reassign the processors to the unexecuted portion of the unfinished tasks according to the above rule. These are

1. A task is completed.
2. We reach a point where, if we were to continue the present assignment, we would be executing some tasks at a lower level at a faster rate than other tasks at a higher level.
3. New tasks arrive.

Algorithm D is essentially the Muntz-Coffman algorithm, except that another new event — when new tasks arrive — is added. In Sections 3.1 and 3.2, we will give the theorems that Algorithm D is an optimal offline algorithm for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$  and

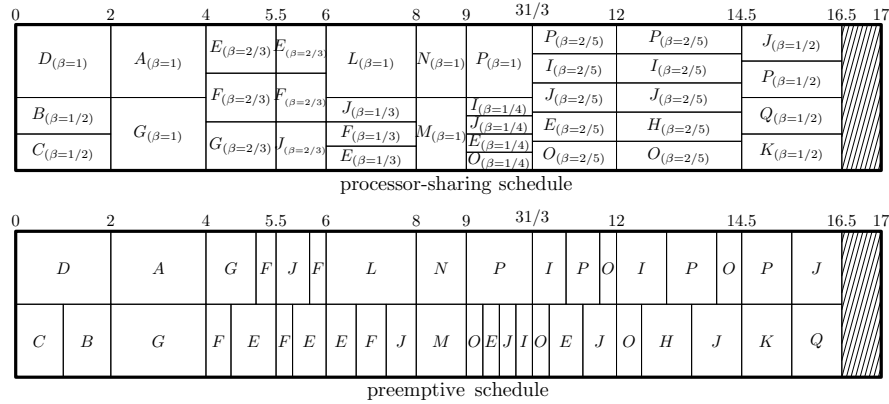


Figure 10. Example illustrating Algorithm D

$P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$ , respectively. Shown in Fig. 10 are the processor-sharing schedule and the preemptive schedule constructed by Algorithm D for the instance given in Fig. 9.

Our proofs that Algorithm C is optimal for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$  and  $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$  are based on the fact that Algorithm D is an optimal offline algorithm for these two cases.

### 3.1 Arbitrary precedence constraint and two processors

We first show that Algorithm D is an optimal offline algorithm for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$ , and then show that Algorithm C is an optimal online algorithm for the same case.

**LEMMA 9** *Let  $S$  be the processor-sharing schedule produced by the Muntz-Coffman algorithm for an instance of  $P2 \mid pmtn, prec \mid C_{\max}$ . Then,  $S$  has the minimum idle processor time at any time instant  $t$ .*

Using the same technique as in Theorem 5 and the property given in Lemma 9, we can show that Algorithm D is an optimal offline algorithm for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$ . This is stated in the next theorem whose proof we will omit.

**THEOREM 10** *Algorithm D is an optimal offline algorithm for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$ .*

**THEOREM 11** *Algorithm C is an optimal online algorithm for  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$ .*

### 3.2 Outtrees and arbitrary number of processors

We first prove that Algorithm D is an optimal offline algorithm for  $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$ , and then prove that Algorithm C is an optimal online algorithm for the same case.

**THEOREM 12** *Algorithm D is an optimal offline algorithm for  $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$ .*

**THEOREM 13** *Algorithm C is an optimal online algorithm for  $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$ .*



## 4. Conclusion

In this article we have given optimal online algorithms for the problems:

- (1)  $P2 \mid p_j = 1, prec_i \text{ released at } r_i \mid C_{\max}$ .
- (2)  $P \mid p_j = 1, outtree_i \text{ released at } r_i \mid C_{\max}$ .
- (3)  $P2 \mid pmtn, prec_i \text{ released at } r_i \mid C_{\max}$ .
- (4)  $P \mid pmtn, outtree_i \text{ released at } r_i \mid C_{\max}$ .

We also show that it is impossible to have optimal online algorithms for the problems:

- (1)  $P3 \mid p_j = 1, intree_i \text{ released at } r_i \mid C_{\max}$ .
- (2)  $P2 \mid p_j = p, chains_i \text{ released at } r_i \mid C_{\max}$ .
- (3)  $P3 \mid pmtn, p_j = 1, intree_i \text{ released at } r_i \mid C_{\max}$ .

Instead of the makespan objective, one wonders whether there are optimal online algorithms for the mean flow time objective. In this regard, it can be shown that Algorithm A is an optimal online algorithm for  $P2 \mid p_j = 1, prec_i \text{ released at } r_i \mid \sum C_j$ , while Algorithm B is an optimal online algorithm for  $P \mid p_j = 1, outtree_i \text{ released at } r_i \mid \sum C_j$ . The proof in Theorem 1 also shows that it is impossible to have optimal online algorithms for  $P3 \mid p_j = 1, intree_i \text{ released at } r_i \mid \sum C_j$  and  $P3 \mid pmtn, p_j = 1, intree_i \text{ released at } r_i \mid \sum C_j$ . Relatively little is known about preemptive scheduling. For example, is it possible to have an optimal online algorithm for  $P2 \mid pmtn, p_j = 1, prec_i \text{ released at } r_i \mid \sum C_j$ ? Recently, Coffman *et al.* [2] gave an algorithm that simultaneously minimizes the makespan and the mean flow time for  $P2 \mid pmtn, p_j = 1, prec_i \text{ released at } r_i \mid \sum C_j$ . Is it possible to adapt their algorithm to yield an optimal online algorithm for this case?

## References

- [1] E. G. Coffman, Jr. and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1 (1972), pp. 200–213.
- [2] E. G. Coffman, Jr., J. Sethuraman, and V. G. Timkovsky. Ideal preemptive schedules on two processors. *Acta Informatica*, 39 (2003), pp. 597–612.
- [3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.
- [4] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Math.*, 5 (1979), pp. 287–326.
- [5] K. S. Hong and J. Y-T. Leung. On-line scheduling of real-time tasks. *IEEE Transactions on Computers*, C41 (1992), pp. 1326–1331.
- [6] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9 (1961), pp. 841–848.
- [7] R. McNaughton. Scheduling with deadlines and loss functions *Management Science*, 6 (1959), pp. 1–12.
- [8] R. R. Muntz and E. G. Coffman, Jr. Optimal preemptive scheduling on two-processor systems *IEEE Transactions on Computers*, C-18 (1969), pp. 1014–1020.
- [9] R. R. Muntz and E. G. Coffman, Jr. Preemptive scheduling of real-time tasks on multiprocessor systems *J. of the Association for Computing Machinery*, 17 (1970), pp. 324–338.

# BICRITERIA PARALLEL TASK SCHEDULING

Fabien Baille\*,<sup>1</sup> Evripidis Bampis,<sup>1</sup> Christian Laforest,<sup>1</sup> Christophe Rapine.<sup>2</sup>

<sup>1</sup>*LaMI, CNRS UMR 8042, Université d'Evry, Tour Evry 2, 523, Place des Terrasses 91000 Evry, France*

*{fbaille,bampis,laforest}@lami.univ-evry.fr*

<sup>2</sup>*Laboratoire GILCO, 46 avenue Félix Viallet, 38031 Grenoble Cedex 1, France*  
*rapine@gilco.inpg.fr*

**Abstract** In this paper, we study the problem of scheduling on  $k$  identical machines a set of parallel tasks with release dates and deadlines in order to optimize simultaneously two conflicting criteria, namely the *Size* (number of scheduled tasks) and the *Weight* (sum of the weight of scheduled tasks). We distinguish two variants of the problem: In the first one, tasks have to be scheduled in a contiguous subset of machines, whereas in the second one, they can be scheduled on any subset of machines. In both variants, we show that if the maximal number of required machines is greater than  $\frac{k}{2}$  then there is no  $(a, b)$ -approximate schedule for any two constants  $a$  and  $b$ . Nevertheless, if the number of required machines is no more than  $\frac{k}{2}$  then we propose a  $(3\alpha, 3\beta)$ -approximation algorithm for the contiguous case and a  $(6\alpha, 6\beta)$ -approximation for the non-contiguous case, where  $\alpha$  (resp.  $\beta$ ) denotes the approximation ratio of a monocriterion algorithm maximizing the *Size* (resp. the *Weight*).

**Keywords:** Multiprocessor Scheduling, Bicriteria approximation

## 1. Introduction - Notations

In this paper, we study the problem of scheduling parallel tasks on a set of  $k$  identical machines. Tasks may require more than a single machine to be executed. They are also subject to release dates and deadlines, representing a time window during which the task has to be scheduled. Each task is associated to a positive weight  $w_i$ . We search for the schedule maximizing simultaneously the number of scheduled tasks and their weights. Clearly, the scheduler may reject tasks if this rejection leads to a better global performance for the considered criteria. This problem models various practical situations in which a set of potential users (clients) requires the resources of a system (e.g. the processors of a parallel computer, the communication channels of a telecommunication

\*Corresponding author.

network, etc) for a prespecified amount of time within a given time window. These resources are managed by a provider which has to maximize simultaneously the number of satisfied clients (number of scheduled tasks) and its own profit (sum of the weights). It is clear that there is a need of tradeoff solutions.

More formally, let  $M$  be a set of  $k \geq 1$  identical machines  $M_1, \dots, M_k$ . Let  $\sigma_i = (r_i, p_i, d_i, \gamma_i, w_i)$  be a *parallel task* where  $r_i$  denotes the *release date*,  $p_i$  the *processing time*,  $d_i$  the *deadline*,  $\gamma_i$  the *number of required machines*, and  $w_i$  the *weight or profit*. All these parameters are positive,  $\gamma_i$  is an integer satisfying  $1 \leq \gamma_i \leq k$  and  $d_i \geq r_i + p_i$ . A task is said to be *sequential* if  $\gamma_i = 1$ , otherwise it is said to be *parallel*. If  $d_i = r_i + p_i$ , the task is called *interval*. An instance  $I$  of our problem is then completely described by the couple  $(\sigma, k)$  where  $\sigma$  denotes the set of parallel tasks and  $k$  the number of machines.

A task  $\sigma_i$  is said to be *accepted* or *scheduled* if  $\gamma_i$  machines are devoted *simultaneously* and without interruption (preemption is not allowed) to its execution in the time window  $[r_i, d_i]$  for a time  $p_i$  (i.e.  $\gamma_i$  machines become simultaneously busy for a time  $p_i$ ). At each date, a machine can execute at most *one* task. A schedule is said to be *valid* if the above conditions are satisfied for all tasks appearing on it (not all tasks need to be scheduled). We distinguish two variants of the problem: We call the case where every task has to be scheduled on a bloc of  $\gamma_i$  consecutive machines, i.e. the case where the indices of the  $\gamma_i$  machines devoted to the execution of  $\sigma_i$  are consecutive, the *contiguous* case. Otherwise, we talk about the *non-contiguous* case.

Two objectives are considered in this paper: The maximization of the number of scheduled tasks, called **NUMBER** or **SIZE** problem in the sequel, and the maximization of the sum of the weights of scheduled tasks, called **WEIGHT** or **PROFIT** problem in the sequel. If  $O$  denotes a valid schedule of an instance given by a set of tasks  $\sigma$  and  $k$  identical machines, then we denote by  $N(O)$  (resp. by  $W(O)$ ) the number of tasks accepted by  $O$  (resp. the sum of the weights of the tasks accepted by  $O$ ) and  $N^*(\sigma)$  (resp.  $W^*(\sigma)$ ) the number (resp. the weight) of the tasks accepted by an optimal schedule of  $\sigma$  on  $k$  machines.

Both problems are NP-hard in the general case [Bar-Noy et al., 2001, Bar-Noy et al., 1999], thus, we try to find good approximate solutions. As usual, we consider the notion of *approximation ratio*. For a criterion  $C$  ( $C = N$  or  $W$ ), an algorithm  $A$  is a  $\rho$ -*approximation algorithm* if the following inequality is satisfied for every instance  $\sigma$ :

$$\rho C(A(\sigma)) \geq C^*(\sigma),$$

where  $A(\sigma)$  denotes the schedule obtained by  $A$  on instance  $\sigma$ . Then,  $\rho$  is called the *approximation ratio* of  $A$ . A schedule is said to be an  $(\alpha, \beta)$ -approximated schedule if it is a factor  $\alpha$  away from the optimum of the first criterion and a factor of  $\beta$  away from the optimum of the second one. If an

algorithm returns such a solution for all instances, it is said to be an  $(\alpha, \beta)$ -approximation algorithm. In what follows, the first element of the couple will denote the approximation ratio for the *Size* and the second will represent the approximation ratio for the *Weight* criterion.

It is not hard to see that, whereas the *SIZE* problem is a subproblem of the *WEIGHT* problem (by letting all the weights to be equal to 1), the two criteria are conflicting (for example, take an instance composed by a single task requiring  $k$  machines for its execution and having a huge weight and a big number of sequential tasks with small weights that have to be scheduled in the same time interval as the  $k$  machine task). Clearly a good solution for one criterion may lead to a poor solution for the other. An alternative is to find (when possible) a solution that simultaneously guarantees good approximation ratios for the two criteria. This is the goal of this paper.

## Related works

For the sequential interval case ( $\gamma_i = 1, \forall i = 1, 2, \dots, n$ ), the two monocriteria problems are solvable in polynomial time (for the *SIZE* see [Carlisle and Lloyd, 1995, Faigle and Nawijn, 1995, Faigle and Nawijn, 1991], while for the *WEIGHT* see [Arkin and Silverberg, 1987, Carlisle and Lloyd, 1995]). For the sequential bicriteria interval case a  $(\frac{k}{r}, \frac{k}{k-r})$ -approximation algorithm ( $1 \leq r < k$ ) has been proposed in [Baille et al., 2004b]. Note that these cases are special cases of what [Spieksma, 1999] called the job interval scheduling problem, where each job consists in a set of intervals and a job is scheduled when one of its intervals is scheduled.

For the sequential task case, the corresponding monocriteria problems are NP-hard and can be approximated within a factor of 1.58 (for the *SIZE*) and 2.54 (for the *WEIGHT*) ([Bar-Noy et al., 1999]). For the bicriteria case, there is a  $(1.58\frac{k}{r}, 2.54\frac{k}{k-r})$ -approximation algorithm ([Baille et al., 2004b]).

For the contiguous parallel interval case, the two monocriteria problems are NP-hard and there is a  $1.58\frac{1}{1-\varepsilon}$ -approximation algorithm for the *WEIGHT* (and thus for the *SIZE* problem) for any fixed  $0 \leq \varepsilon < 1$  and bounded  $\gamma_i$ 's, see [Chen et al., 2002].

For the non-contiguous parallel intervals case, the *WEIGHT* problem is NP-hard and can be approximated within a factor of 3 (see [Calinescu et al., 2002]). In the same paper, it is also shown that the special case where  $\max\{\gamma_i\} \leq \frac{k}{2}$  is approximable within a factor 2. As far as we know, the NP-hardness of the *SIZE* problem is still open. Nevertheless, since the *SIZE* is a subproblem of the *WEIGHT* problem, the previous result shows that the *SIZE* problem is also approximable within a factor of 3 (and within 2 when  $\max\{\gamma_i\} \leq \frac{k}{2}$ ).

For the contiguous parallel tasks case, the monocriteria problems are NP-hard and approximable within a factor of 35 (see [Bar-Noy et al., 2001]).

For the non-contiguous parallel tasks case, the monocriteria problems are NP-hard and there is a 4-approximation proposed in [Bar-Noy et al., 2001] for the WEIGHT (and thus for the SIZE problem).

The contribution of this paper concerns the bicriteria approach of the two last variants mentioned above. When  $\max\{\gamma_i\} \leq \frac{k}{2}$ , we give existence theorems of approximated bicriteria schedules implying  $(\alpha, \beta)$ -approximation algorithms for the contiguous (in Section 2.1) and for the non-contiguous (in Section 2.2) cases. In Section 3 we exhibit non reachable couples of approximation ratios, when  $\max\{\gamma_i\} > \frac{k}{2}$ .

## 2. Existence of approximate schedules and approximation algorithms

In this section, we prove that there exists approximated schedules for all the instances where  $\max\{\gamma_i\} \leq \frac{k}{2}$ . We consider the contiguous case in Section 2.1, and the non-contiguous case in Section 2.2. In both cases the general idea is the following: We start with two schedules on  $k$  machines, each one optimal for one of the two criteria. Then, we extract from these schedules 3 sub-schedules on  $\frac{k}{2}$  machines (assume for simplicity that  $k$  is even), and we choose among them two sub-schedules namely the sub-schedule with maximal *Size* and the sub-schedule with maximal *Weight*. Finally, we merge in an appropriate way the chosen sub-schedules in order to obtain a valid schedule on  $k$  machines.

### 2.1 Contiguous case

#### Existence of a (3, 3)-approximate schedule.

**THEOREM 1** *For every instances  $I = (\sigma, k)$  where  $\max\{\gamma_i\} \leq \lfloor \frac{k}{2} \rfloor$  and  $k \geq 3$ , there is a (3, 3)-approximated contiguous schedule.*

**Proof.** Let  $O_N^*$  be a contiguous schedule of  $\sigma$  accepting an optimal number of tasks ( $N(O_N^*) = N^*(\sigma)$ ). Let  $O_W^*$  be a contiguous schedule of  $\sigma$  with an optimal sum of weights of accepted tasks ( $W(O_W^*) = W^*(\sigma)$ ).

In  $O_N^*$ , consider the machine  $M_{\lceil \frac{k}{2} \rceil}$ . Three valid contiguous sub-schedules of the tasks accepted by  $O_N^*$  on  $\lfloor \frac{k}{2} \rfloor$  machines can be distinguished:

- The first one consists in the set of tasks exclusively scheduled on the  $(\lceil \frac{k}{2} \rceil - 1)$  first machines. Since  $\lceil \frac{k}{2} \rceil - 1 \leq \lfloor \frac{k}{2} \rfloor$ , all these contiguous tasks can be scheduled on  $\lfloor \frac{k}{2} \rfloor$  machines at the same dates and at the same machines as in  $O_N^*$ .
- The second one consists in the set of tasks exclusively scheduled in  $O_N^*$  on the set of machines from  $M_{\lceil \frac{k}{2} \rceil + 1}$  to  $M_k$ . Using similar arguments

as those used for the first set, we can show that all these tasks can be scheduled on  $\lfloor \frac{k}{2} \rfloor$  machines.

- The third one consists in the set of tasks using machine  $M_{\lceil \frac{k}{2} \rceil}$ . All these tasks can be scheduled on  $\lfloor \frac{k}{2} \rfloor$  machines. Indeed, it is sufficient to schedule each task at the same dates as in  $O_N^*$ . The obtained schedule is feasible since  $\max\{\gamma_i\} \leq \lfloor \frac{k}{2} \rfloor$  and since there is no two tasks executed at the same date ( $O_N^*$  is valid thus  $M_{\lceil \frac{k}{2} \rceil}$  - and so the sub-schedule - accepts at most one task per date).

Let  $\mathcal{N}$  be the schedule on  $\lfloor \frac{k}{2} \rfloor$  machines among the 3 above schedules containing the maximum number of tasks. Since all these three schedules form a partition of  $O_N^*$ , we get  $3N(\mathcal{N}) \geq N(O_N^*)$ .

By partitioning  $O_W^*$  in the same manner, we get a schedule  $\mathcal{W}$  on  $\lfloor \frac{k}{2} \rfloor$  machines satisfying  $3W(\mathcal{W}) \geq W(O_W^*)$ .

Let  $O$  be the schedule composed by  $\mathcal{N}$  on the  $\lfloor \frac{k}{2} \rfloor$  first machines, in which all tasks belonging to  $\mathcal{N} \cap \mathcal{W}$  have been deleted, and by  $\mathcal{W}$  in the  $\lfloor \frac{k}{2} \rfloor$  next machines. Note that  $O$  is a contiguous schedule of  $\sigma$  and that when  $k$  is odd, there is at least an idle machine during the whole schedule.

Since all the tasks deleted from  $\mathcal{N}$  are in  $\mathcal{W}$ , we get  $3N(O) \geq N(O_N^*)$ , and  $3W(O) \geq W(O_W^*)$  follows from  $W(O) \geq W(\mathcal{W})$ . The proposed schedule  $O$  is then a  $(3, 3)$ -approximate contiguous schedule of  $\sigma$  on  $k$  machines.  $\square$

**Approximation algorithm.** As a consequence of Theorem 1, we show in this section how to construct in polynomial time a  $(3\alpha, 3\beta)$ -approximation schedule, mixing an  $\alpha$ -approximation algorithm for the SIZE problem and a  $\beta$ -approximation algorithm for the WEIGHT problem. As a corollary, we obtain an algorithm with a couple of approximation ratios of  $(4.74 \frac{1}{1-\epsilon}, 4.74 \frac{1}{1-\epsilon})$  for the contiguous parallel intervals variant and a  $(105, 105)$ -approximation for the contiguous parallel tasks variant.

**COROLLARY 1** *Given an  $\alpha$ -approximation algorithm for the contiguous SIZE problem and a  $\beta$ -approximation algorithm for the contiguous WEIGHT problem, there is a  $(3\alpha, 3\beta)$ -approximation algorithm for the contiguous bicriteria problem on a system of  $k \geq 3$  machines.*

**Proof.** Let  $A_N$  be the  $\alpha$ -approximation algorithm for the SIZE and  $A_W$  be the  $\beta$ -approximation algorithm for the WEIGHT (both for the contiguous case) problem.

Let  $O$  be the schedule returned by the algorithm presented in the proof of Theorem 1 with the only difference that now we start with  $A_N(\sigma)$  and  $A_W(\sigma)$ , the schedules respectively returned by  $A_N$  and  $A_W$ . Clearly, the algorithm

can be executed in polynomial time.  $O$  is a valid and contiguous schedule. Thus, with similar arguments as those used in the proof of Theorem 1, we get:  $3N(O) \geq N(A_N(\sigma))$  and  $3W(O) \geq W(A_W(\sigma))$ . Since  $A_N$  is an  $\alpha$ -approximation algorithm and  $A_W$  is a  $\beta$ -approximation algorithm, we have by definition:  $\alpha N(A_N(\sigma)) \geq N^*(\sigma)$  and  $\beta W(A_W(\sigma)) \geq W^*(\sigma)$ . Therefore,  $O$  verifies simultaneously:  $3\alpha N(O) \geq N^*(\sigma)$  and  $3\beta W(O) \geq W^*(\sigma)$ .  $\square$

**COROLLARY 2** *For the contiguous interval scheduling problem there is an algorithm yielding a couple of approximation ratios of  $(4.74\frac{1}{1-\varepsilon}, 4.74\frac{1}{1-\varepsilon})$ , for all  $\varepsilon > 0$  on a system of  $k \geq 3$  machines.*

**Proof.** It is a consequence of Corollary 1 using the algorithm of [Chen et al., 2002] ( $\alpha = \beta = 1.58\frac{1}{1-\varepsilon}$ ).  $\square$

**COROLLARY 3** *There exists a  $(105, 105)$ -approximation algorithm, for the contiguous task scheduling problem on a system of  $k \geq 3$  machines.*

**Proof.** It is a consequence of Corollary 1 using the algorithm of [Bar-Noy et al., 2001] ( $\alpha = \beta = 35$ ).  $\square$

## 2.2 Non-contiguous case

In the previous section, we have shown that all contiguous schedules on  $k$  machines can be partitionned into three sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines. In this section, we want to use the same idea as the previous section in order to obtain a bicriteria approximation algorithm for the non-contiguous case. However, we cannot apply directly the proof of theorem 1. If we try to do it, since a task can be scheduled on any set of machines, a situation in which a “piece” of task appears in each sub-schedules can occur. That is why we describe, in this section, a specific algorithm which takes into account the non-contiguity of the tasks.

**Existence of a  $(6, 6)$ -approximate schedule.** In this section, we show the existence of a  $(6, 6)$ -approximate schedule for the non-contiguous variant of the problem. To obtain this result, we show how to partition a schedule on  $k$  machines into 6 sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines. We consider two cases: in the first one we treat instances in which every task requires at most  $\frac{k}{4}$  machines for its execution, while in the second case every task requires more than  $\frac{k}{4}$  machines but less than  $\lfloor \frac{k}{2} \rfloor$  machines.

**LEMMA 1** *For every instance  $I = (\sigma, k)$  where  $\max\{\gamma_i\} \leq \frac{k}{4}$  and  $k \geq 4$ , the set of accepted tasks by every schedule  $O$  of  $\sigma$  on  $k$  machines can be partitioned into 3 valid sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines.*

**Proof.**

In order to prove this lemma, we present the following greedy algorithm that returns a partition of  $O$  into sub-schedules each one using at most  $\lfloor \frac{k}{2} \rfloor$  machines. Then, we show that the number of sub-schedules is at most three.

We need the following notations. We denote by  $O_{\lfloor \frac{k}{2} \rfloor}^j$  the  $j$ -th sub-schedule of  $O$  on  $\lfloor \frac{k}{2} \rfloor$  machines constructed by our algorithm. We also denote by  $l$  the total number of constructed sub-schedules. Given a schedule  $O$ , we construct an instance  $\sigma'$  of intervals in the following way: We replace each task  $\sigma_i = (r_i, p_i, d_i, \gamma_i, w_i)$  scheduled between its starting date  $s_i$  and its ending date  $e_i$  in  $O$  by the interval  $\sigma'_i = (r'_i = s_i, p'_i = p_i, d'_i = e_i, \gamma'_i = \gamma_i, w'_i = w_i)$ .

Here is the description of our greedy algorithm.

- $l = 1$ .
- For each  $\sigma'_i$  of  $\sigma'$  taken in the non-decreasing order of the starting dates  $s_i$ , do:
  - Find (if it exists) the smallest  $j$  ( $j \in \{1, \dots, l\}$ ) such that  $\sigma_i$  can be scheduled into  $O_{\lfloor \frac{k}{2} \rfloor}^j$ .
  - If  $\sigma'_i$  cannot be included into one of the  $l$  existing sub-schedules, then schedule  $\sigma'_i$  on a new schedule  $O_{\lfloor \frac{k}{2} \rfloor}^{l+1}$  and set  $l = l + 1$ .

Clearly, this algorithm outputs  $l$  sub-schedules of  $O$  on  $\lfloor \frac{k}{2} \rfloor$  machines. By construction, all these sub-schedules are valid and form a partition of  $O$ . Note also that, by construction of  $\sigma'$ , each task is executed at the same time interval as in  $O$ .

**Claim:**

For every schedule  $O$  of non-contiguous tasks, the greedy algorithm partitions  $O$  into 3 sub-schedules, i.e.,  $l \leq 3$ .

**Proof of the claim:** By contradiction, suppose that  $O_{\lfloor \frac{k}{2} \rfloor}^4$  is non-empty. Let  $\sigma_i$  be the first task scheduled in  $O_{\lfloor \frac{k}{2} \rfloor}^4$ . Let  $s_i$  and  $e_i$  be respectively the starting and ending dates of task  $\sigma_i$  in  $O$ . Let us consider  $O_{\lfloor \frac{k}{2} \rfloor}^1$ ,  $O_{\lfloor \frac{k}{2} \rfloor}^2$ , and  $O_{\lfloor \frac{k}{2} \rfloor}^3$  at the time  $s_i$  where the algorithm considers  $\sigma_i$  for inclusion.

Since  $\sigma_i$  cannot be scheduled neither in  $O_{\lfloor \frac{k}{2} \rfloor}^1$  nor in  $O_{\lfloor \frac{k}{2} \rfloor}^2$  nor in  $O_{\lfloor \frac{k}{2} \rfloor}^3$  and since the tasks are treated in non-decreasing order of their release dates,  $s_i$  is the date at which the number of busy machines is maximal in  $O_{\lfloor \frac{k}{2} \rfloor}^1$ ,  $O_{\lfloor \frac{k}{2} \rfloor}^2$  and  $O_{\lfloor \frac{k}{2} \rfloor}^3$ .



At date  $s_i$ , the number of busy machines in each of the schedules  $O_{\lfloor \frac{k}{2} \rfloor}^1$ ,  $O_{\lfloor \frac{k}{2} \rfloor}^2$ , and  $O_{\lfloor \frac{k}{2} \rfloor}^3$  is at least  $\lfloor \frac{k}{2} \rfloor - \gamma_i + 1$  (otherwise,  $\sigma_i$  could be added to one of the above schedules by the algorithm). Since  $O_{\lfloor \frac{k}{2} \rfloor}^1$ ,  $O_{\lfloor \frac{k}{2} \rfloor}^2$ , and  $O_{\lfloor \frac{k}{2} \rfloor}^3$  are disjoint by construction, and since the total number of busy machines in  $O$  is at most  $k$ , we must have at date  $s_i$ :

$$3 \left( \left\lfloor \frac{k}{2} \right\rfloor - \gamma_i + 1 \right) + \gamma_i \leq k$$

thus

$$\gamma_i \geq \frac{1}{2} \left( 3 \left\lfloor \frac{k}{2} \right\rfloor - k + 3 \right).$$

Since  $k \leq \lfloor \frac{k}{2} \rfloor + \lfloor \frac{k}{2} \rfloor + 1$ , we get

$$\gamma_i \geq \frac{1}{2} \left( 3 \left\lfloor \frac{k}{2} \right\rfloor - \left\lfloor \frac{k}{2} \right\rfloor - \left\lfloor \frac{k}{2} \right\rfloor - 1 + 3 \right)$$

so,

$$\gamma_i \geq \frac{1}{2} \left\lfloor \frac{k}{2} \right\rfloor + 1 \tag{1}$$

If  $k$  is even, then there is an integer  $p$  such that  $k = 2p$  and the inequality (1) becomes:

$$\gamma_i \geq \frac{1}{2} \left\lfloor \frac{2p}{2} \right\rfloor + 1 = \frac{p}{2} + 1 = \frac{k}{4} + 1$$

This is in contradiction with the hypothesis:  $\max\{\gamma_i\} \leq \frac{k}{4}$ .

If  $k$  is odd, then there is an integer  $p$  such that  $k = 2p + 1$  and (1) becomes:

$$\gamma_i \geq \frac{1}{2} \left\lfloor \frac{2p+1}{2} \right\rfloor + 1 = \frac{1}{2} \left( p + \left\lfloor \frac{1}{2} \right\rfloor \right) + 1 \geq \frac{p}{2} + 1$$

Since  $\frac{k}{4} = \frac{2p+1}{4} = \frac{p}{2} + \frac{1}{4}$ , we have  $\frac{p}{2} + 1 > \frac{p}{2} + \frac{1}{4}$  thus

$$\gamma_i > \frac{k}{4}$$

This also contradicts the fact that  $\max\{\gamma_i\} \leq \frac{k}{4}$ . This proves the claim that proves the Lemma.  $\square$

**LEMMA 2** *For every instance  $I = (\sigma, k)$  for which  $\lfloor \frac{k}{2} \rfloor \geq \max\{\gamma_i\}$  and  $\min\{\gamma_i\} > \frac{k}{4}$  and  $k \geq 2$ , the set of accepted tasks by every schedule  $O$  of  $\sigma$  on  $k$  machines can be partitioned into 3 sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines.*

**Proof.** Since  $\frac{k}{4} + 1$  is the smallest value for a  $\gamma_i$ , at most 3 tasks are scheduled by the  $k$  machines at any date.

Thus, by partitioning  $O$  with the greedy algorithm of the proof of Lemma 1, we can show that the fourth sub-schedule is necessarily empty, otherwise, the hypothesis of the validity of  $O$  is violated.  $\square$

**THEOREM 2** *For every instance  $I = (\sigma, k)$  satisfying  $\max\{\gamma_i\} \leq \lfloor \frac{k}{2} \rfloor$  and  $k \geq 4$ , there exists a  $(6, 6)$ -approximated schedule.*

**Proof.** Let  $O_N^*$  (resp.  $O_W^*$ ) be an optimal schedule of  $\sigma$  on  $k$  machines for the *Number* (resp. the *Weight*) criterion. We partition the tasks scheduled by  $O_N^*$  (and  $O_W^*$ ) into *big* and *small* tasks. A task is called big (resp. small) if  $\gamma_i > \frac{k}{4}$  (resp.  $\gamma_i \leq \frac{k}{4}$ ).

Let  $S_N^*$  (resp.  $S_W^*$ ) be the schedule on  $k$  machines resulting from the deletion of big tasks from  $O_N^*$  (resp.  $O_W^*$ ) and let  $B_N^*$  (resp.  $B_W^*$ ) be the schedule on  $k$  machines resulting from the deletion of small tasks from  $O_N^*$  (resp.  $O_W^*$ ).

By construction,  $B_N^* \cup S_N^* = O_N^*$  and  $B_N^* \cap S_N^* = \emptyset$ . For the same reasons  $B_W^*$  and  $S_W^*$  form a partition of  $O_W^*$ .

Since  $B_N^*$  and  $B_W^*$  contain, by construction, only tasks such that  $\gamma_i > \frac{k}{4}$  and since  $\max\{\gamma_i\} \leq \lfloor \frac{k}{2} \rfloor$ , by Lemma 2, we can partition them into 3 sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines.

Since  $S_N^*$  and  $S_W^*$  only contain tasks such that  $\gamma_i \leq \frac{k}{4}$ , by Lemma 1 we can partition them into 3 sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines.

Thus, we have partitioned both  $O_N^*$  and  $O_W^*$  into 6 sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines. Let  $\mathcal{M}_N$  be the sub-schedule of the partition of  $O_N^*$  with the maximum *Size*. Let  $\mathcal{M}_W$  be the sub-schedule of the partition of  $O_W^*$  with the maximum *Weight*.

We have  $6N(\mathcal{M}_N) \geq N(O_N^*)$  and  $6W(\mathcal{M}_W) \geq W(O_W^*)$ .

Let  $O$  be the (valid) schedule on  $k$  machines executing on the  $\lfloor \frac{k}{2} \rfloor$  first machines the tasks of  $\mathcal{M}_W$  at the same dates as in  $O_W^*$  and the tasks of  $\mathcal{M}_N - \mathcal{M}_W$  on the  $\lfloor \frac{k}{2} \rfloor$  next machines.

Clearly, we have  $6W(O) \geq W(O_W^*)$  because  $W(\mathcal{M}_W) \leq W(O)$ .

Since the tasks contained in  $\mathcal{M}_W \cap \mathcal{M}_N$  appear also in  $\mathcal{M}_W$ , each task of  $\mathcal{M}_N$  appears in  $O$ . Thus  $N(\mathcal{M}_N) \leq N(O)$  and  $6N(O) \geq N(O_N^*)$ .  $\square$

**Approximation algorithm for the non-contiguous case.** We show here that applying the same partitioning as the one described in the proof of Theorem 2 starting from two approximate schedules (instead of two optimal ones) implies that a bicriteria approximate schedule can be computed in polynomial time.

**COROLLARY 4** *Given an instance  $I = (\sigma, k)$  with  $k \geq 4$ , an  $\alpha$ -approximation algorithm for the non-contiguous SIZE problem and a  $\beta$ -approximation algorithm for the non-contiguous WEIGHT problem, a polynomial time  $(6\alpha, 6\beta)$ -approximation algorithm for the bicriteria non-contiguous problem exists.*

**Proof.** Let  $A_N$  be the  $\alpha$ -approximation algorithm for the SIZE problem and  $A_W$  be the  $\beta$ -approximation algorithm for the WEIGHT problem.

- 1 Partition (in the same manner as mentioned in Lemmas 1 and 2) both  $A_N(\sigma)$  and  $A_W(\sigma)$  into 6 sub-schedules on  $\lfloor \frac{k}{2} \rfloor$  machines.
- 2 Choose the schedule having the maximum Size among the sub-schedules of the partition of  $A_N(\sigma)$  and the schedule having the maximum Weight among the sub-schedules in the partition of  $A_W(\sigma)$ .
- 3 Merge (i.e. make the union) these two schedules deleting redundancies.

Let  $O$  be the schedule returned by the above algorithm. Using similar arguments as those used in the proof of Theorem 2, we easily get:  $6N(O) \geq N(A_N(\sigma))$  and  $6W(O) \geq W(A_W(\sigma))$ . Thus, since algorithm  $A_N$  is an  $\alpha$ -approximation algorithm and  $A_W$  is a  $\beta$ -approximation algorithm, we have by definition:  $\alpha N(A_N(\sigma)) \geq N^*(\sigma)$  and  $\beta W(A_W(\sigma)) \geq W^*(\sigma)$ . Therefore,  $O$  verifies simultaneously:  $6\alpha N(O) \geq N^*(\sigma)$  and  $6\beta W(O) \geq W^*(\sigma)$ . Note that the algorithm can be executed in polynomial time.  $\square$

**COROLLARY 5** *For the non-contiguous parallel interval scheduling problem with  $k \geq 4$ , there is an algorithm yielding a couple of approximation ratios of  $(12, 12)$ .*

**Proof.** It is an application of Corollary 4, using the 2-approximation algorithm for the WEIGHT problem described in [Calinescu et al., 2002] ( $\alpha = \beta = 2$ ).  $\square$

**COROLLARY 6** *There exists an algorithm yielding a couple of approximation ratios of  $(24, 24)$  for the non-contiguous parallel task scheduling problem with  $k \geq 4$ .*

**Proof.** It is an application of Corollary 4, using the 4-approximation algorithm for the WEIGHT problem described in [Bar-Noy et al., 2001]. ( $\alpha = \beta = 4$ ).  $\square$

### 3. Negative results

In the previous section, we have analyzed the case where a task could not require more than half of the total number of machines. In this section, we show that when there is no restriction on the number of required machines, some couples of approximation ratios are unreachable.

**THEOREM 3** *If the number of required machines  $\gamma_i$  is greater than  $\frac{k}{2}$  then for all constants  $\alpha$  and  $\beta$ , there exists instances  $I = (\sigma, k)$  with  $k \geq 3$  for which there is no  $(\alpha, \beta)$ -approximation schedule.*

**Proof.** Let us consider  $k > 2$  machines and the following set of intervals depending on an integer  $\mathcal{W}$ . All the intervals of the instance require  $\lceil \frac{k}{2} \rceil + 1$  machines. They are of two types:

- One long interval ( $r_1 = 0, p_1 = \mathcal{W}^2, d_1 = \mathcal{W}^2, w_1 = \mathcal{W}^2$ ).
- $\mathcal{W}$  short intervals ( $r_i = i, p_i = 1, d_i = i + 1, w_i = 1$ ) for  $i = 2, \dots, \mathcal{W} + 1$ .

We have  $N^*(\sigma) = \mathcal{W}$  (schedule all the shorts intervals), and  $W^*(\sigma) = \mathcal{W}^2$  (schedule the unique long interval). Thus, an optimal schedule for the SIZE problem has a *weight*  $\mathcal{W}$  away from the optimal. Conversely, an optimal schedule for the WEIGHT problem has a *size*  $\mathcal{W}$  away from the optimal. Since  $\mathcal{W}$  is unbounded, and since the short intervals intersect the long ones, any constant approximated schedule for one metric can be as far as possible from the optimum for the other metric. This proves the theorem.  $\square$

### 4. Conclusion and future work

In this paper, we have studied the problem of scheduling parallel tasks in order to optimize two conflicting criteria namely the *Size* and the *Weight*. When the maximum number of required machines is no more than  $\frac{k}{2}$ , we proposed approximation results for the contiguous and the non-contiguous variants of the problem. When  $\gamma_i > \frac{k}{2}$ , we show that there is no  $(\alpha, \beta)$ -approximate schedule for any couple of fixed constants  $\alpha$  and  $\beta$ . It is interesting to know whether it is possible to improve the result on the non-contiguous case by obtaining the same couple of approximation ratios as in the contiguous case. Furthermore, the extension of our approach to other task models like the degradable processing time model (see [Baille et al., 2004a]) and malleable task model (see [Mounie et al., 1999]) should be examined in the considered bicriteria model.

### References

- [Arkin and Silverberg, 1987] Arkin, E. and Silverberg, B. (1987). Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8.

- [Baille et al., 2004a] Baille, F., Bampis, E., and Laforest, C. (2004a). Maximization of the size and the weight of schedules of degradable intervals. In Chwa, K.-Y. and Munro, I., editors, *proceedings of COCOON'04*, LNCS No. 3106, pages 219–228. Springer-Verlag.
- [Baille et al., 2004b] Baille, F., Bampis, E., and Laforest, C. (2004b). A note on bicriteria schedules with optimal approximation ratios. *Parallel Processing Letters*, 14:315–323.
- [Bar-Noy et al., 2001] Bar-Noy, A., Bar-Yehuda, R., Freund, A., (Seffi) Naor, J., and Schieber, B. (2001). A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090.
- [Bar-Noy et al., 1999] Bar-Noy, A., Guha, S., Naor, J., and Schieber, B. (1999). Approximating the throughput of multiple machines under real-time scheduling. In *STOCS 99*, pages 622–631.
- [Calinescu et al., 2002] Calinescu, G., Amit, C., Karloff, H., and Rabani, Y. (2002). Improved approximation algorithms for resource allocation. In Springer-Verlag, editor, *Lecture Notes in Computer Science 2337 - IPCO, the 9th Conference on Integer Programming and Combinatorial Optimization*, pages 401–414.
- [Carlisle and Lloyd, 1995] Carlisle, M. C. and Lloyd, E. L. (1995). On the  $k$ -coloring of intervals. *Discrete Applied Mathematics*, 59:225–235.
- [Chen et al., 2002] Chen, B., Hassin, R., and Tzur, M. (2002). Allocation of bandwidth and storage. *IIE Transaction*, 34:501–507.
- [Faigle and Nawijn, 1995] Faigle, U. and Nawijn, M. (1995). Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58:13–17.
- [Faigle and Nawijn, 1991] Faigle, U. and Nawijn, W. (1991). Greedy  $k$ -decomposition of intervals order. In *Proceedings 2nd Twente Workshop on Graphs and Combinatorial Optimization*, pages 53–56. Department of Applied Mathematics, University of Twente.
- [Mounie et al., 1999] Mounie, G., Rapine, C., and Trystram, D. (1999). Efficient approximation algorithms for scheduling malleable tasks. In *SPAA '99: Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures*, pages 23–32. ACM Press.
- [Spieksma, 1999] Spieksma, F. (1999). On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227.

# ON PROJECT SCHEDULING WITH AND/OR PRECEDENCE CONSTRAINTS

George Adelson-Velsky

*Holon Academic Institute of Technology, Holon, Israel*

Alexander Gelbukh

*National Polytechnical Institute, Mexico City, Mexico*

Eugene Levner

*Holon Academic Institute of Technology, Holon, Israel*

**Abstract:** We address a PERT/CPM scheduling problem with AND/OR constraints and examine its relations with extremal problems in grammars and hypergraphs. We demonstrate that two scheduling algorithms developed independently by Adelson-Velsky *et al.* (2002) and Mohring *et al.* (2004) have different worst-case complexity and, in a sense, are incomparable.

**Key words:** PERT/CPM, AND/OR constraints, AND/OR graphs

## 1. INTRODUCTION

In this paper, we address a PERT/CPM scheduling problem with AND/OR constraints [1–5, 8–10] and study its relationships with two other extremal problems which historically have been investigated independently: a problem of finding a min-value string in a context-free grammar [7] and a path-finding problem in hypergraphs [6]. Although the three problems are quite different, we show that in non-trivial special cases they become pair-wise equivalent (and solvable by Dijkstra-type greedy algorithms). Next, we compare two known PERT/CPM scheduling problems with AND/OR constraints and show them to be special cases of a more general well-solvable problem. We demonstrate that two algorithms developed independently by Adelson-Velsky *et al.* [1–4] and Mohring *et al.* [8–10] have different worst-case complexity and, in a sense, are incomparable; their hypergraph interpretation permits to further expose their distinction.

The paper is organized as follows. In Section 2 we present the problems. In Section 3 we study the relationships between the problems. In Section 4 we compare two known PERT/CPM project scheduling problems with AND/OR constraints and show that two earlier algorithms have different worst-case complexity. This section contains, also, a hypergraph interpretation of the two problems. Section 5 summarizes our study and outlines some new directions for future research.

## 2. PROBLEM DESCRIPTION

In this section, we present the formulations of three extremal problems under consideration.

### 2.1 Problem 1: PERT/CPM with AND/OR Constraints

Consider graphs with two types of nodes called *AND-nodes* and *OR-nodes*. The AND-nodes are standard nodes used in the PERT/CPM project models: for any operation corresponding to an AND-node, all its immediately preceding operations must be completed before this operation starts. Another type of nodes, OR-nodes, models a situation when an operation can start as soon as at least one of its immediately preceding operations is completed; such an operation is represented by an OR-node.

The input to the problem under consideration is  $\langle G, s, \tau \rangle$ , where  $G = (V, E)$  is a directed graph with nodes of both types,  $V$  is the node set,  $|V| = n$ ,  $E$  is the arc set,  $|E| = p$ ,  $\tau$  is an arc length, and  $s$  is a selected node, called the *start*, whose occurrence time is given, say  $t(s) = 0$ .

Consider the following system of inequalities:

$$t(v_j) \geq t(s) \quad \text{for all } v_j, \quad (1)$$

$$t(v_j) \geq \max_{v_i \in P(v_j)} (t(v_i) + \tau(v_i, v_j)) \quad \text{if } v_j \in A, \quad (2)$$

$$t(v_j) \geq \min_{v_i \in P(v_j)} (t(v_i) + \tau(v_i, v_j)) = \min_{v_i \in P(v_j)} t(v_i) \quad \text{if } v_j \in O. \quad (3)$$

Here  $P(v_j)$  denotes the set of nodes predecessors to  $v_j$ .

*The PERT/CPM scheduling with AND-OR constraints:* Find the earliest occurrence times  $t^*(v)$  for each vertex  $v \in V$ , satisfying the conditions (1)–(3).

The problem is a generalization of the critical path problem and the shortest path problem: it turns into the former one if there are only OR-nodes in the graph and into the latter one if there are only AND-nodes. Further, as we will see below, this problem includes, as special cases, the PERT/CPM scheduling problems with AND/OR conditions considered in [1–3] and [10].

### 2.2 Problem 2: Finding Optimal Strings in Grammars

In 1977, Knuth [7] introduced a class of context-free grammars, called *superior grammars*, and developed a fast algorithm for solving an extremal problem of finding a min-value terminal string on such grammars.

**Definition 1.** Let  $\mathbf{R}_\infty$  be the set of the nonnegative real numbers extended with the value  $+\infty$ . A function  $g(x_1, \dots, x_k)$  from  $\mathbf{R}^k$  into  $\mathbf{R}_\infty$  is called a *superior function*, if it is monotone non-decreasing in each variable and  $g(x_1, \dots, x_k) \geq \max(x_1, \dots, x_k)$  for all  $x_1, \dots, x_k$ . A context-free grammar is called *superior*, if all its productions are of the general form:

$$Y \rightarrow g(X_1, \dots, X_k),$$

where  $Y, X_1, \dots, X_k$  are non-terminal symbols,  $g$  is a terminal symbol corresponding to a superior function, and “(”, “)”, and “,” are also terminal symbols; when  $k = 0$ , “ $g$ ” is a terminal symbol (called a *terminal literal*), corresponding to a non-negative real constant.

Let  $L(Y) = \{\alpha \mid \alpha \in T^* \text{ and } Y \rightarrow^* \alpha\}$  be the set of terminal strings, over the terminal alphabet  $T$ , derivable from  $Y$ .

Every string  $\alpha$  in  $L(Y)$  is a composition of superior functions. Since the terminal literals correspond to given non-negative real constants,  $\alpha$  corresponds to a uniquely defined non-negative real number denoted  $val(\alpha)$ .

The *Knuth Problem* (KP): Find a terminal string  $\alpha$  of minimal value,  $m^*(Y) = \min \{val(\alpha) \mid \alpha \in L(Y)\}$ , for each non-terminal  $Y$ .

Example 1 [7]. Consider the following production rules for non-terminals  $A$ ,  $B$ , and  $C$ , and terminal literals  $a$  and  $e$ :

$$\begin{aligned} A &\rightarrow a, & B &\rightarrow c(A), & C &\rightarrow e, \\ A &\rightarrow b(B, C), & B &\rightarrow d(A, C, A), & C &\rightarrow f(B, A). \end{aligned}$$

Let  $a = 4$ ;  $b(x, y) = \max(x, y)$ ;  $d(x, y, z) = x + \max(y, z)$ ;  $c(x) = x + 1$ ;  $e = 9$ ;  $f(x, y) = \frac{1}{2}(x + y + \max(x, y))$ . Then  $L(A) = \{a; b(c(a), e); b(c(a), f(c(a), a)); b(c(b(c(a), e)), e); \dots\}$ , with the values  $val(\alpha)$  being 4, 9, 7, 10, etc. The minimal value here is 4, which is achieved at string  $a$ .

### 2.3 Problem 3: Finding Paths in Hypergraphs

Definition 2. A *hypergraph* is a pair  $H = (V, E)$ , where  $V$  is the set of nodes and  $E = \{E_i\}$  the set of *hyperedges*. A directed hyperedge, or *hyperarc*, is an ordered pair  $E_i = (X_i, Y_i)$  of disjoint subsets of nodes.

Definition 3 [6]. A *path* in a hypergraph is a sequence of nodes and adjacent hyperarcs. In a *simple* path all hyperarcs are distinct. A path is a *cycle*, if its final node belongs to the set of tails of the first hyperarc in the path.

Definition 4 [6]. Given a hypergraph  $H = (V, E)$ , a *B-path* from  $s$  to  $t$  is a minimal hypergraph  $H_P = (V_P, E_P)$  such that:

- (i)  $E_P \subseteq E$ ;
- (ii)  $s, t \in V_P = \bigcup_{E_i \in E_P} (X_i \cup Y_i) \subseteq V$ ;
- (iii)  $x \in V_P \Rightarrow x$  is connected in  $H_P$  to node  $s$  by means of a cycle-free simple path.

The idea behind the latter concept is to provide an instrument for studying connections between start node  $s$  and other nodes in hypergraphs (see [6] for more details).

Definition 5 [6]. Given a B-path from  $s$  to  $t$ , a *length function*  $W(y)$  assigns lengths to each its node as follows:  $W(s) = 0$ ;  $W(y) = \min_E \{w(E) + F(T(E))\}$ , where  $w(E)$  is a given real length function on hyperarcs,  $E$  are the hyperarcs of the B-path entering  $y$ , and  $F$  is a non-decreasing function of  $W(x)$ , for each  $x \in T(E)$ ; here  $T(E)$  is the set of tails of  $E$ . *Length* of the B-path is the length assigned to final node  $t$ .

*The Min-length B-path Problem*: Find a min-length B-path on hypergraphs with a non-decreasing length function  $F$ .

Three problems presented in this section have numerous applications in computer science, artificial intelligence, robotics, and scheduling of real-time systems (see [1–5, 10] and numerous examples and references therein).

## 3. PROBLEM REDUCTION

Problems 1 and 2 above can be reformulated in hypergraph terms. Indeed, any AND/OR graph may be considered as a special type hypergraph, in which OR-nodes of the AND/OR graph correspond to the nodes of the hypergraph while each AND-node together with its in-arcs and out-arcs can be looked at as a hyperarc in the hypergraph (with the MAX-operator assigned to this hyperarc). Further, any terminal and non-terminal symbol in a grammar may be graphically represented as a node in the hypergraph while a production rule is represented as a hyperarc.



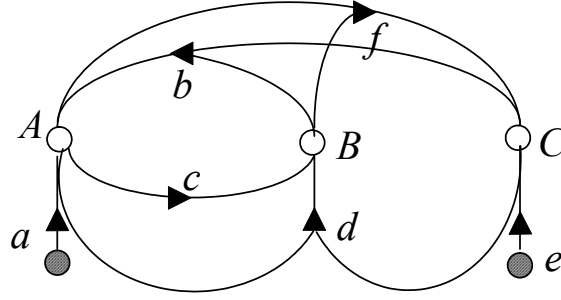


Figure 1. A sample hypergraph

Proposition. Problems 1–3 have different optimal solutions.

We will illustrate this claim using the following example.

Example 2. Consider a hypergraph presented in Fig. 1. In this example, a hypergraph (and a corresponding AND/OR graph) is presented by the following conditions:

$$A \leftarrow a, B \leftarrow d(A, C), C \leftarrow b(B), C \leftarrow c(B, A), C \leftarrow e;$$

where “ $X \leftarrow$ ” denotes a hyperarc entering a node  $X$ , lowercase letters denote arc lengths, and  $a = 1$ ,  $e = 20$ ,  $b = x$ ,  $d = \max(x+1, y)$ ,  $c = \max(x+2, y)$ .

For this example, the optimal solution to the Knuth problem is:

$$\alpha(A) = a, \alpha(B) = d(a, e), \alpha(C) = e; m(A) = 1, m(B) = m(C) = 20.$$

In this example, the optimal solution to the min-length B-path problem is: optimal B-path to A is  $a$ ; optimal B-path to B is  $a-c-d$ ; optimal B-path to C is  $a-c$ ; their lengths are:  $l(A) = 1$ ,  $l(B) = l(C) = 3$ .

Finally, the optimal routes in the AND-OR graph are:  $p(A) = \{a\}$ ,  $p(B) = \{a-d\}$ ;  $p(C) = \{a-c\}$ . The earliest times are:  $t(A) = 1$ ,  $t(B) = t(C) = 2$ .

These solutions are presented graphically in Fig. 2.

Now we show that although in general the three problems have different solutions, in non-trivial special cases they become pair-wise equivalent (and solvable by Dijkstra-type greedy algorithms).

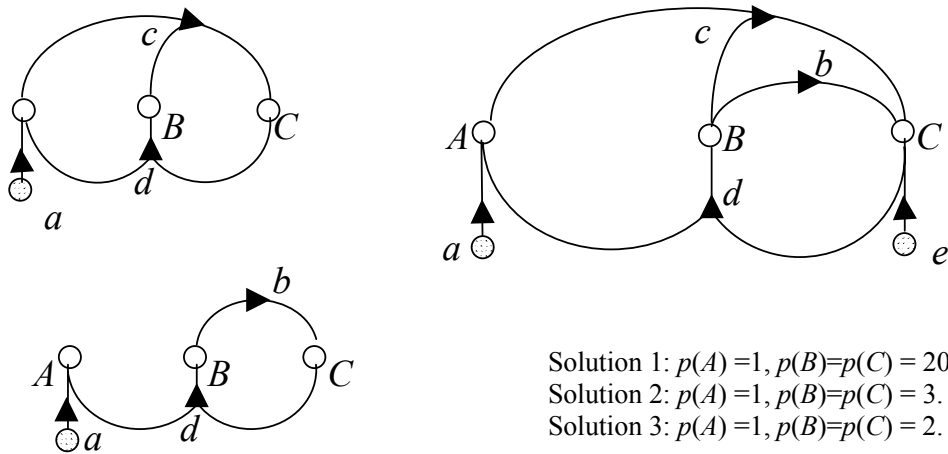


Figure 2. Optimal solutions to the problems

**Definition 6.** A terminal string in the Knuth grammar is called *legal* if it does not contain repeating substrings and cycles.

**Theorem (The Equivalence Theorem).** The PERT/CPM scheduling problem (“problem 1”) with no zero-length cycles is equivalent to the problem of finding a legal min-value terminal string in the Knuth superior grammars, which, in turn, is equivalent to finding a min-weight acyclic B-path in hypergraphs with superior weights induced by the corresponding superior grammar.

The proof follows straightforwardly from problems’ definitions and is skipped here. Theorem can be extended to a wider class of context-free grammars which we call *strongly monotone* (which is a weaker condition than Knuth’s superiority), see [4].

## 4. COMPARISON OF TWO PERT/CPM PROBLEMS

Rolf Mohring, Martin Skutella and Frederik Stork [8–10] have investigated and elegantly solved a project scheduling problem with AND/OR precedence constraints. A similar problem has been studied by Adelson-Velsky *et al.* [1–4], who have given different motivations and algorithms. In this section we explicitly expose distinctions between the two problems and show that they are two distinct cases of a more general (and well-solvable) scheduling problem 1. We show that two algorithms developed independently in [3] and [10], in spite of their apparent similarity, have different worst-case complexity and, in a sense, are incomparable. More exactly, each algorithm is designed to solve “its own” problem only. We present a hypergraph interpretation of the two problems which vividly demonstrates that the relation between the problems is essentially the same as that between the shortest-path problem and critical-path problem in standard graphs: whereas the problem in [1-4] can be reformulated as a problem of finding a hyperpath of *minimum* total length, its counterpart [8-10] can be viewed as a problem of finding a hyperpath of *maximum* total length.

Recall that for a given set  $V$  of jobs, a precedence constraint of AND-type represents the requirement that a job can only start after the completion of all its predecessors; such a job is represented by an AND-node. Similarly, a precedence constraint of OR-type claims that a job cannot start before some of its predecessors is finished; such a job is represented by an OR-node. As in [1-5, 8-10], we consider a representation of the scheduling problem by a directed bipartite graph  $D$ , in which a node-set  $N = V \cup W$ , where  $V$  is a set of AND-nodes, and  $W$  a set of OR-nodes. Each arc  $(i, j)$  represents a precedence relations between jobs  $i$  and  $j$ ;  $A$  denotes the set of all arcs. The sets  $V$  and  $W$  form a bipartition of  $D$ . Node  $s$  is a selected start node.

We intend to compare the following scheduling problems introduced in [1-4] and [8-10]:

**Problem A** ([1-4]). For each arc  $(i, j)$  leading to a job  $j$  of AND-type we are given a non-negative *time lag*  $t(v_i, v_j)$ . All other arcs (leading to OR-nodes) have zero lengths. The problem is to find a component-wise minimal schedule (i.e., a vector of earliest start times)  $S = (S(v_1), \dots, S(v_n))$ , where  $n = |N|$ ,  $N = V \cup W$ , satisfying the following conditions:

$$S(s) = 0, \tag{4}$$

$$S(v_j) \geq \max_{v_i \in P(v_j)} (S(v_i) + t(v_i, v_j)) \quad \text{if } v_j \in V, \tag{5}$$

$$S(v_j) \geq \min_{v_i \in P(v_j)} S(v_i) \quad \text{if } v_j \in W. \tag{6}$$

Here  $P(v)$  denotes the set of nodes – immediate predecessors to node  $v$ .

**Problem B** [8-10]. For each OR-node  $w$  there are one or more entering arcs  $(i, w)$ ,  $i \in V$ , and a single outcoming arc,  $(w, j)$ . For each arc  $(i, j)$  leading to an OR-node  $j$  we are given a non-negative time lag  $d(v_i, v_j)$ . All other arcs (leading to AND-nodes) have zero lengths. The problem is to find a component-

wise minimal schedule (i.e., a vector of earliest start times)  $S = (S(v_1), \dots, S(v_n))$ , where  $n = |N|$ ,  $N = V \cup W$ , satisfying the following conditions:

$$S(s) = 0, \quad (7)$$

$$S(v_j) \geq \max_{v_i \in P(v_j)} S(v_i) \quad \text{if } v_j \in V, \quad (8)$$

$$S(v_j) \geq \min_{v_i \in P(v_j)} (S(v_i) + d(v_i, v_j)) \quad \text{if } v_j \in W. \quad (9)$$

## 4.1 Algorithms

Adelson-Velsky and Levner [1, 2] have observed that known greedy path-finding algorithms fail to solve Problem A and discovered a non-greedy algorithm of complexity  $O(|A|^2)$  for that problem. This algorithm has been improved in [3] to solve the problem in  $O(|A| \times |V|)$  time. Independently, another algorithm has been developed by Mohring *et al.* [8-10] for solving Problem B; it uses a complementary structure and runs in  $O(|A| \times |W|)$  time. These two facts may misleadingly give us an impression that a combined algorithm of complexity  $O(|A| \times \min(|W|, |V|))$  could be readily constructed for the both problems. However, as we will see below, this is not the case.

In order to simplify the further presentation, we will rephrase the earlier algorithms [1-4, 8-10] using a more perceptible language based on the concept of *black* or *red node-painting*.

*Node Painting.* At each iteration, the algorithm in [3] paint each node *red* whenever it discovers that the start time of the node is strictly greater than the current time  $t$ . On the other hand, the algorithms paint a node *black* if its minimum start time is found. The algorithms start with  $S_s = 0$  (so,  $s$  is *black*). As the algorithms proceed, the set of black nodes grows at least by one per iteration. The algorithms stop when all the optimal start times are found.

We will use the following terminology:

- The *front*  $F$  of  $D$  is the set of bAND-nodes in  $D$  adjacent to start  $s$ .
- The body  $B$  of  $D$  is  $U \setminus (F \cup s)$ .
- The *maximum node* in  $F$  is an AND-node  $v^*$  with  $t(s, v^*) = \max \{t(s, v_j) \mid v_j \in F\}$ .
- A node is *planned* if it assigned a minimum start time;  $U$  denotes the set of planned nodes.

The algorithm for Problem B in [8-10] (further denoted as MSS) is the following:

---

**Algorithm MSS:** Computing earliest start times in Problem B.

---

**Input:** A digraph  $D$  with non-negative arc weights on the arcs in  $V \times W$  and zero-length arcs in  $W \times V$ .

**Output:** Optimal schedule  $S$  for Problem B.

0. Initialization: set  $t := 0$ ; all  $S_i := \infty$ ;

remove all arcs with positive lengths  $d(v_i, v_j)$  from  $D$ .

1. **do**

2. recursively paint *black* each AND-node  $j$  if all its predecessors are *black* or  $P(j)$  is empty, and paint each OR-node  $w$  black if at least one of its predecessors is *black*;

3. all *black* nodes are planned to be started at the current time  $t$ ;

4. whenever an AND-node  $j$  is planned, the start times of all neighboring OR-nodes  $w$  following  $j$  in  $D$  (i.e. such that  $(j, w) \in D$ ) are updated to

$$S(w) := \min (S(w), S(j) + d(j, w)).$$

5. choose the OR-node with  $\min S(w)$ ; paint it *black*; set  $t = \min S(w)$ .

6. **until** no OR-node is left unpainted

7. **return**  $S$ .

---

The worst-case running time of the MSS is  $O(|A| \times |W|)$ . Indeed, the number of iterations is at most  $|W|$ , as at least one OR-node is planned per iteration, and the number of operations within each iteration is at most  $O(|A|)$ . We refer to [4] and [6], for the complete proofs and more details.

The scheme of the algorithm [3] for solving problem  $A$ , here denoted as AVGL, is presented below.

---

**Algorithm AVGL:** Computing earliest start times in Problem A.

**Input:** A digraph  $D$  with non-negative arc weights on the arcs in  $W \times V$  and zero-length arcs in  $V \times W$ .

**Output:** Optimal schedule  $S$  for Problem A.

0. Initialization:  $t := 0$ ;  $S_s := 0$  (start nodes  $s$  is planned at time 0); all other  $S_i := \infty$ .
  1. **do** (in following *red*-painting operations, *black* nodes and their adjacent arcs are not taken into account)
  2.   First red painting: paint *red* all AND-nodes in the body  $B$  of  $G$ ,  $v_j \in V \cap B$ , such that there exists at least one arc  $(v_i, v_j)$  with  $t(v_i, v_j) > 0$ ;  
       remove all the arcs  $(v_i, v_j)$  with  $t(v_i, v_j) > 0$ ;
  3.   Second red painting: paint *red* any OR-node in  $B$  if all its immediate predecessors are already *red*;  
       paint *red* any AND-node in  $B$  if any one of its predecessors is painted *red*;
  4.   **do**
  5.     Third red painting: paint *red* the maximum unpainted node  $v^*$  in  $F$ ;  
       then recursively paint *red* other unpainted nodes in  $F$  using the rules in line 3;
  6.     if all nodes in  $F$  are painted *red* then
  7.       all nodes just painted red at step 5 are planned to have their earliest start time  $S_i$  at time  $t(s, v^*)$   
       and are painted *black*;  
       update the times of neighboring nodes the same way as in MSS [3]; current time  $t := t(s, v^*)$
  8.     **until** no node in  $F$  is left unpainted;
  9.   **until** no AND-node is left unpainted,
  10. **return**  $S$ .
- 

The worst-case running time of the AVGL is, obviously,  $O(|A| \times |V|)$ . Indeed, the number of iterations is at most  $|V|$  since the number of planned AND-nodes grows at least by one at each iteration. The number of operations within each iteration is  $O(|A|)$  because each arc is scanned at rows 1 to 7 at most once. As for finding the maximum element in  $F$ , in row 4, we initially sort  $F$  in  $O(|V| \times \log |V|)$  time, all further adjustments of the sorted list being done in at most  $\log |V|$  operations per iteration. We refer to [1] and [3] for complete proofs and more details.

The MSS algorithm looks more elegant and lucid in comparison with the AVGL; this is because the problem structure is different, and at least one OR-node in Problem B can be straightforwardly planned at each iteration without using the multi-stage red painting.

We can see that the algorithm AVGL designed for Problem A is inapplicable for solving Problem B, and vice versa. In other words, these algorithms are applicable to different and complementary problems. In this sense, the two algorithms are incomparable.

## 4.2 A Hypergraph Interpretation

In this section we return to the hypergraph representation of the PERT/CPM problem introduced in Section 3. Informally speaking, a B-hyperpath  $p$  in a hypergraph  $G$  is a sub-hypergraph such that if a node of a hypergraph  $G$  is in  $p$  then all its predecessors in  $G$  must also be in  $p$ . In Problem A, each OR-node is represented by a node of a hypergraph while each AND-node is substituted by a hyperarc with the max-operator  $\max_{v_i \in P(v_j)} (S(v_i) + t(v_i, v_j))$  being assigned to the hyperarc as its “length”.

The zero-length arc in Problem A leading from each OR-node may be naturally interpreted as a “head-part” of the hyperarc going to a corresponding head-node of the hyperarc. Then value  $S(v_j)$  defined by (5) in Problem A may be looked at the “length” of a hyperpath leading from start  $s$  to  $v_j$ , whereas

Problem A is interpreted as that of finding the *min-length* hyper-paths leading from the start to all other nodes in the hypergraph  $G$ .

Reciprocally, in Problem B we represent each AND-node in the AND-OR graph as a node of a hypergraph, and each OR-node as a hyperarc with the min-operator  $\min_{v_i \in P(v_j)} (S(v_i) + d(v_i, v_j))$  assigned to each hyperarc.

The zero-length arc in Problem B is again interpreted as a "head-part" of the hyperarc going to a head-node in the hypergraph. Now the length of a hyperpath is defined by Eq. (9), and we wish to find the *max-length* hyperpaths between the start and all other nodes in the hypergraph.

The relation between the two problems is essentially the same as that between the shortest-path problem and critical-path problem in standard graphs. Once more, we can see that the corresponding algorithms [3] and [10] are designed and applied to quite different and complementary problems, and, as such, are incomparable.

## 5. CONCLUDING REMARKS

This study has been inspired by the algorithms in [3, 7, 8, 11]. A question motivating our work has been to understand why several seemingly different problems can be solved by very similar greedy Dijkstra-type algorithms. The present study demonstrates that although the three problems are quite different, each one has a well-solvable special case where those problems are pair-wise equivalent: Knuth's problem on superior grammars is reducible to a corresponding problem in superior-length hypergraphs (and vice versa), and the two problems are equivalent to the PERT/CPM scheduling problem in AND-OR graphs without zero-length cycles.

Obviously, Problems A and B of Section 6 can be seen as special cases of the PERT/CPM scheduling problem 1 of Section 2. Notice that this problem has been formulated in [3] on a general (not-bipartite) AND/OR graph and solved with the complexity of  $O(|A| \times n) = O(|A| \times (|V| + |W|))$ ; the corresponding algorithm being a generalization of the above algorithm AVGL. Apparently, algorithm MSS can also be generalized to solve Problem 3 – by adding a corresponding *red* painting step that has been introduced in AVGL. However, this rough transformation leads to the worst-case complexity of  $O(|A| \times n)$ , and we do not know if a faster development of the MSS does exist. A challenging open question is to derive an algorithm of complexity  $O(|A| \times \min(|W|, |V|))$ , or prove that this is impossible unless  $P = NP$ .

In the future, it would be interesting to find an algorithm capable of handling negative arc lengths in AND-OR graphs in polynomial time. Another possible area is to explore a deep connection between the considered problems and logic, in particular, propositional satisfiability.

**Acknowledgements.** The work was done under partial support of Mexican Government (SNI, CONACyT, IPN) to the second author.

## REFERENCES

- [1] G. Adelson-Velsky and Levner, E., *Project scheduling in AND-OR graphs: A generalization of Dijkstra's algorithm*, Technical Report, Department of Computer Science, Holon Academic Institute of Technology, Holon, Israel, 1999.
- [2] G. Adelson-Velsky and Levner, E., Project scheduling in AND-OR graphs: A generalization of Dijkstra's algorithm, *Mathematics of Operations Research*, 27 (3), (2002) 504–517.
- [3] G. Adelson-Velsky, A. Gelbukh, and E. Levner, On fast path-finding algorithms in AND-OR graphs. *J. Mathematical Problems in Engineering*, 8(4/5), (2002) 283–293.
- [4] G. Adelson-Velsky, E. Levner, and A. Gelbukh, Analysis of extremal problems on grammars, hypergraphs, and AND-OR graphs based on generalized Bellman equations. In S. Sahni (ed.), *Proceedings of the IASTED Conference on Advances in*

- Computer Science and Technology*, November 22024, 2004, St. Thomas, US Virgin Islands, ACTA Press, Anaheim, 2004, 111–116.
- [5] Y. Desmedt and Y. Wang, Vulnerability of complex systems and critical vertices in AND/OR graphs, *Int. Journal of Foundations of Computer Science*, 2004.
  - [6] G. Gallo, G. Longo, S. Pallotino, S. Nguen, Directed hypergraphs and applications, *Discrete Applied Mathematics*, 42, 1993, 177–201.
  - [7] D. E. Knuth, A generalization of Dijkstra’s algorithm, *Information Processing Letters*, 6(1), 1977, 1–5.
  - [8] R.H. Mohring, Skutella, M. and Stork, F. *Forcing relations for AND/OR Precedence Constraints*, in Proceedings of the XI Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’00), New York, Philadelphia, 2000, pp. 235-236.
  - [9] R.H. Mohring, Skutella, M. and Stork, F. *Scheduling with AND/OR Precedence Constraints*, Technical Report No. 689/2000, Technische Universitat Berlin, August 2000, 26 pp.
  - [10] R.H. Mohring, Skutella, M. and Stork, F. Scheduling with AND/OR Precedence Constraints, *SIAM J. on Computing*, 33(2), 2004, pp. 393–415.

# NEW TIGHT NP-HARDNESS OF PREEMPTIVE MULTIPROCESSOR AND OPEN SHOP SCHEDULING

Evgeny Shchepin  
*Steklov Math. Inst., 117966  
Gubkina 8, Moscow, Russia  
scep@mian.ras.su*

Nodari Vakhania  
*Science Faculty, State University of Morelos  
Av. Universidad 1001, Cuernavaca 62210, Morelos Mexico.  
Inst. Computational Math., Akuri 8, Tbilisi 93, Georgia  
nodari@servm.fc.uaem.mx*

**Abstract** We show that preemptive versions of some NP-hard scheduling problems remain NP-hard, but only for a restricted number of preemptions. If we allow a “sufficient” number of preemptions, then these problems become polynomially solvable. We find, as we call, the critical number of preemptions for our problems, i.e., the minimal number of preemptions for which they become polynomially solvable. First we establish that the critical number of preemptions for scheduling  $m$  identical processors is  $m - 1$  (we show that scheduling identical processors with at most  $m - 2$  preemptions is NP-hard). Then we consider so-called acyclic open-shop scheduling problem with  $m$  machines, a strongly restricted version of the classical open-shop scheduling, and show that  $m - 2$  is the critical number of preemptions for this problem (the earlier known related well-known result was that non-preemptive open shop scheduling is NP-hard). Finally, we consider a slightly restricted version of scheduling on  $m$  unrelated processors. The restriction is that the processing time of any job on any processor is no more than the optimal schedule makespan. We call such processors non-lazy unrelated processors. We show that  $2m - 3$  is the critical number of preemptions for scheduling non-lazy unrelated processors.

**Keywords:** open-shop scheduling, multiprocessor scheduling, preemption, algorithm, NP-hard

## Introduction

Two types of scheduling problems distinguished in the literature are preemptive and non-preemptive ones. It is quite typical that a non-preemptive scheduling problem  $\mathcal{P}$  is NP-hard whereas its preemptive version  $\mathcal{P}_{pmtn}$  is polynomially solvable. Let us call a scheduling problem  $\pi$ -preemptive if at most  $\pi$  preemptions are allowed in it,  $\pi$  being any non-negative integer. A question which naturally arises is whether  $k$ -preemptive and  $l$ -preemptive problems, for some non-negative integers  $k$  and  $l$ , have the same complexity. If a 0-preemptive (non-preemptive) problem is NP-hard, does the corresponding  $k$ -preemptive problem, for  $k = 1, 2, \dots$ , remain NP-hard? For many scheduling problems this is not the case, i.e., there is some positive integer  $\pi$  such that corresponding  $k$ -preemptive versions, for  $k \geq \pi$ , are polynomially solvable. Then what is the minimal  $\pi$  such that the  $\pi$ -preemptive version is polynomially solvable (we call such  $\pi$  the *critical number* of preemptions for the problem)? Traditionally, a preemptive scheduling problem implies an arbitrary number of preemptions: a  $k$ -preemptive problem for a fixed integer  $k > 0$  would not be treated as a preemptive scheduling problem (unless  $k$  is “sufficiently” large magnitude not known in advance and changing from problem to problem). So though a  $k$ -preemptive problem,  $k > 0$ , is not a non-preemptive problem, it might neither be a “preemptive problem”. We could refine the rough classification of scheduling problems into preemptive or non-preemptive ones by specifying the *maximal number of preemptions* which is allowed for a particular problem, i.e., consider  $\pi$ -preemptive scheduling problems.

A polynomial-time algorithm for a preemptive problem  $\mathcal{P}_{pmtn}$  imposes a certain number of preemptions, the maximal number of which can be usually estimated. In most of the real-life problems the number of preemptions is a crucial factor which has to be made as small as possible since each preemption implies additional communication cost and may also yield a forced job migration. How small the overall number of preemptions can be made, i.e., what is the critical number of preemptions for  $\mathcal{P}_{pmtn}$ ? This kind of question has been also posed by Shachnai et al. [7] and the critical number of preemptions  $2m - 1$  for scheduling uniform machines  $Q/pmtn/C_{\max}$  was established (an  $O(n + m \log m)$  algorithm by Gonzalez and Sahni [3] yields at most  $(2m - 2)$  preemptions). In this paper, we first establish that  $m - 1$  is the critical number of preemptions for scheduling identical machines  $P/pmtn/C_{\max}$ . We show that  $(m - 2)$ -preemptive scheduling on identical processors  $P/pmtn(m - 2)/C_{\max}$  is NP-hard (Section 3), whereas a venerable linear-time algorithm by McNaughton [6] yields  $m - 1$  preemptions.

Multiprocessor and open-shop scheduling problems are intimately related with the so-called acyclic distributions (see Shchepin & Vakhania [8]–[11]). An optimal schedule can be obtained in two stages. On the first stage an op-



timal distribution is constructed (a *distribution* assigns jobs or their parts to machines without specifying the start times). Such a distribution can be constructed in polynomial time by linear programming. The preemptions in these distributions can be represented by the so-called *preemption graphs*: in these graphs nodes represent machines and if two nodes are joined by an edge, then the corresponding machines share the same job which labels this edge. An optimal distribution, obtained by linear programming, has at most  $m - 1$  preemptions or equivalently, its preemption graph is acyclic. A distribution or an open-shop (which can be seen as a distribution) is *acyclic*, if its preemption graph is acyclic. As an example, in an acyclic open-shop no two jobs may have two non-dummy operations on the same two machines. Scheduling acyclic distributions turned out to be an efficient tool for the exact solution of  $R/pmtn/C_{\max}$  (see Lawler and Labetoulle [4]) and an approximate solution of its non-preemptive version  $R//C_{\max}$  (see Lenstra et al. [5] and Shchepin & Vakhania [10]). Acyclic shop scheduling problems are also interesting from the other point of view: they can represent “maximal” polynomially solvable cases of the corresponding non-acyclic versions Shchepin & Vakhania [11].

The second problem we deal with is an acyclic  $(m - 3)$ -preemptive open-shop scheduling  $O/acyclic, pmtn(m - 3)/C_{\max}$ . General preemptive open-shop problem  $O/pmtn/C_{\max}$  is well-known to be solvable in polynomial (no worse than  $O(n^4)$ ) time, and its non-preemptive version  $O//C_{\max}$  is NP-hard due to the early classical results by Gonzales & Sahni [2]. We show that  $(m - 3)$ -preemptive open-shop scheduling, even its acyclic version  $O/acyclic, pmtn(m - 3)/C_{\max}$ , remains NP-hard (Section 4). Moreover,  $m - 2$  is the critical number of preemptions due to the recent linear-time algorithm by Shchepin & Vakhania [11] for  $O/acyclic, pmtn(m - 2)/C_{\max}$ .

Scheduling  $m$  unrelated processors  $R/pmtn/C_{\max}$  is much more difficult than scheduling  $m$  identical processors. No polynomial algorithm with a performance ratio less than 1.5 can exist unless  $P = NP$ , see Lenstra et al. [5] (currently best known ratio is  $2 - 1/m$ , see Shchepin & Vakhania [10]). However, it turned out that a slightly restricted version of  $R/pmtn/C_{\max}$  can be efficiently solved. In particular, we consider, as we call, *non-lazy* unrelated processors, in which the processing time of any job on any machine is no more than the optimal schedule makespan  $C_{\max}^*$ ; i.e., there is no machine which is too slow for some job (we abbreviate this problem by  $R/p_{ij} \leq C_{\max}^*/C_{\max}$  adopting the standard notation for scheduling problems). Not only  $R/p_{ij} \leq C_{\max}^*/C_{\max}$  is polynomially solvable, but its  $(2m - 3)$ -preemptive version is polynomially solvable (Section 6), whereas we show that the  $(2m - 4)$ -preemptive version is NP-hard (Section 5). Thus  $2m - 3$  is the critical number of preemptions for  $R/p_{ij} \leq C_{\max}^*/C_{\max}$ . We can guarantee only a near-optimal  $(2m - 3)$ -preemptive schedule for general unrelated processors. The makespan of such a schedule is no more than either

the corresponding non-preemptive schedule makespan or  $\max\{C_{\max}^*, p_{\max}\}$ , where  $p_{\max}$  is the maximal job processing time (Section 6). These results are obtained relatively easily from the earlier results from Shchepin & Vakhania [9].

## 1. Summary of notions and notations

This section contains glossary of notions and notations used further in this paper. The reader may choose to have a brief look on it or skip it at all now and return to it later upon necessity. Not all the introduced notations are widely used in the literature (however, we do find them convenient).

**Jobs and machines.** We will be dealing with  $n$  jobs  $\mathcal{J} = \{J^1, \dots, J^n\}$  and  $m$  machines (processors)  $\mathcal{M} = \{M_1, \dots, M_m\}$ . The *processing time (length)* of job  $J$  on machine  $M$  will be denoted by  $M(J)$ . The maximal job processing time will be denoted by  $p_{\max}$ .

**Distributions and assignments.** A *distribution*  $\delta$  of  $n$  jobs from  $\mathcal{J}$  on  $m$  machines from  $\mathcal{M}$  is a mapping  $\delta: \mathcal{J} \times \mathcal{M} \rightarrow R^+$ , such that  $\sum_{M \in \mathcal{M}} \delta(J, M) = 1$ , for all  $J \in \mathcal{J}$ . The *processing time (or length)* of job  $J$  on a machine  $M$  in distribution  $\delta$  is  $|J|_M^\delta = \delta(J, M)M(J)$ . The *(total) processing time or length* of a job  $J$  in a distribution  $\delta$  is  $|J|^\delta = \sum_{M \in \mathcal{M}} |J|_M^\delta$ . The maximal job length in  $\delta$  is denoted by  $|\delta|^{\max}$ .

$\sum_{J \in \mathcal{J}} |J|_M^\delta$  is called the *load* of machine  $M$  in distribution  $\delta$  and is denoted by  $|M|_\delta$ . The maximal machine load in  $\delta$  or the *makespan* is denoted by  $|\delta|_{\max}$ . A distribution  $\delta$  for  $\mathcal{J}, \mathcal{M}$  with the minimal  $|\delta|_{\max}$  is called an *optimal distribution*. A *uniform distribution* is a distribution, in which all machine loads are equal. The *total processing time in a distribution* is the sum of all machine loads in this distribution.

The *sequential makespan* of  $\delta$ ,  $\|\delta\| = \max\{|\delta|_{\max}, |\delta|^{\max}\}$ . It is clear that  $\|\delta\|$  is a lower bound on the makespan of any feasible schedule for the corresponding open-shop problem; Gonzales & Sahni [2] (see also Lawler & Labetoulle [4]) have shown that  $\|\delta\|$  is achievable in a feasible schedule associated with distribution  $\delta$  (a formal definition of a feasible schedule associated with  $\delta$  is given a bit later in this section).

An *assignment* of jobs of  $\mathcal{J}$  on machines of  $\mathcal{M}$  is a binary relation on  $\mathcal{J} \times \mathcal{M}$ . Any distribution  $\delta$  of  $\mathcal{J}$  on  $\mathcal{M}$  generates an assignment  $\{(J, M) \mid \delta(J, M) > 0\}$ . We will not use any special letter for an assignment, instead, we will use  $\delta(J)$  for  $\{M \in \mathcal{M} \mid \delta(J, M) > 0\}$ .

A distribution  $\delta$  is *non-preemptive* if  $\delta(J, M)$  takes value 0 or 1 for any  $J, M$ . The number of preemptions of job  $J$  in  $\delta$  is  $pr_J(\delta) = |\delta(J)| - 1$ .  $pr(\delta) = \sum_{J \in \mathcal{J}} pr_J(\delta)$  is the (total) number of preemption in  $\delta$ .

**Multiprocessors.** A *multiprocessor* is a triple constituted by the sets of jobs  $\mathcal{J}$  and machines  $\mathcal{M}$  and a *processing time function*  $f$ , a mapping from  $\mathcal{J} \times \mathcal{M}$  to  $R^+$ , where the value of this function for a pair  $J, M$  is the processing requirement of  $J$  on  $M$ . As mentioned above, we use  $M(J)$  for the value of this function for the pair  $J, M$ . For the notational simplicity, a multiprocessor will be denoted by  $\mathcal{J}, \mathcal{M}$  (we omit  $f$ ).

A multiprocessor without any restriction on its processing time function is called a system of *unrelated processors*. In a system of *identical processors*, for each  $P$  and  $Q$  from  $\mathcal{M}$  and for each  $J \in \mathcal{J}$ ,  $P(J) = Q(J)$ .

**Schedules.** A *schedule* indicates which job is in process on each machine at each time moment. Formally, a schedule  $\sigma$  is a subset of  $\mathcal{J} \times \mathcal{M} \times [0, T)$ , for some  $T \geq 0$ .  $(J, M, t) \in \sigma$  signifies that job  $J$  is processed by machine  $M$  at the moment  $t$  in  $\sigma$ .  $T$  is called the *makespan* of  $\sigma$  and is denoted by  $\|\sigma\|$ . For a pair  $J, M \in \mathcal{J} \times \mathcal{M}$ ,  $\sigma(J, M) = \{t \in R^+ \mid (J, M, t) \in \sigma\}$ ;  $\sigma(J, M)$  may contain a single interval, or it can be a multi-interval, i.e., a union of two or more intervals. Each interval is assumed to be a left half-interval. The total length of  $\sigma(J, M)$ ,  $|\sigma(J, M)|$ , is the *processing time* of  $J$  on  $M$  in  $\sigma$ .

A *feasible schedule*  $\sigma$  must satisfy the following conditions:

- (1) For every  $M \in \mathcal{M}$ , the set of jobs  $\sigma(M, t) = \{J \in \mathcal{J} \mid (J, M, t) \in \sigma\}$  contains at most one element;
  - (2) For every  $J \in \mathcal{J}$ , the set of machines  $\sigma(J, t) = \{M \in \mathcal{M} \mid (J, M, t) \in \sigma\}$  contains at most one element;
  - (3) For every  $J \in \mathcal{J}$ ,  $\sum_{M \in \mathcal{M}} \frac{|\sigma(J, M)|}{M(J)} = 1$ ;
- (1) provides that each machine processes at most one job at any time; (2) provides that each job is processed by at most one machine at any time; (3) provides that each job is completely processed.

A feasible schedule with the minimal makespan is called *optimal*.

With each distribution is *associated* an (infinite) set of *feasible schedules* with this distribution. Conversely, with each feasible schedule  $\sigma$  is *associated* a *distribution*  $\delta_\sigma$  defined as follows:

$$\delta_\sigma(J, M) = \frac{|\sigma(J, M)|}{M(J)}.$$

A machine is *idle* at some moment (interval) if it handles no job at that moment (interval). We call a feasible schedule  $\sigma$  *tight*, if no machine is idle from time 0 to its completion time (that is, the completion time of the latest job scheduled on that machine) in  $\sigma$ . A tight non-preemptive schedule, associated with a distribution  $\delta$  can be uniquely given by indicating a sequence of jobs for each machine. The makespan of this (not necessarily feasible) schedule is  $|\delta|_{\max}$ .

**Switching points, components, splittings and preemptions.** Let us define a *component* of a schedule as the maximal (continuous) time interval, dur-

ing which a machine processes a unique job. A schedule can be completely given by all its components. Formally, a component of a schedule  $\sigma$  is a triple  $(J, M, [p, q])$ , where  $J$  is a job,  $M$  is a machine and  $[p, q]$  is a time interval which is a connectivity component in  $\sigma(J, M)$ . Further on we will refer to a component  $(J, M, [p, q])$  as a *J-component of  $\sigma$  on  $M$*  or a *(J, M)-component of  $\sigma$* .

A boundary point of the set  $\sigma(J, M)$  is called a *switching point of  $J$  on  $M$*  or a *J-switching point on  $M$* . At such a point  $M$  interrupts the processing of  $J$  or (re)starts its processing.

We will say that a job  $J$  is *split* on machine  $M$ , if  $\sigma(J, M)$  is a multi-interval, i.e., it consists of two or more  $J$ -components. The *number of splittings* of job  $J$  on machine  $M$  in  $\sigma$ ,  $sp(\sigma(J, M))$  is the number of components in  $\sigma(J, M)$  minus 1.  $sp(\sigma(M)) = \sum_{J \in \mathcal{J}} sp(\sigma(J, M))$  is the number of splittings in  $\sigma$  on  $M$ ;  $sp(\sigma) = \sum_{M \in \mathcal{M}} sp(\sigma(M))$  is the total number of splittings in  $\sigma$ .

The *number of preemptions* in  $\sigma$ ,  $pr(\sigma) = sp(\sigma) + pr(\delta_\sigma)$ : a preemption in  $\sigma$  may come either from  $\delta_\sigma$ , or it might be a splitting.

**Preemption and assignment graphs.** A (full) *preemption graph* of a distribution  $\delta$  is a labelled graph, where the nodes correspond to the machines of  $\mathcal{M}$  and the edges correspond to the jobs of  $\mathcal{J}$ . In particular, there is an edge  $(M_i, M_j)$  between nodes  $M_i$  and  $M_j$  labelled by job  $J$ , iff both  $\delta(J, M_i)$  and  $\delta(J, M_j)$  are positive. The *reduced preemption graph* of  $\delta$  or the *preemption graph* for short, is a subgraph  $G(\delta)$  of the full preemption graph, in which all redundant edges are eliminated: an edge  $(M_i, M_j)$  labelled by  $J$  is *redundant*, if there exists  $k$ ,  $i < k < j$ , such that  $\delta(J, M_k) > 0$  (according to our machine indexing in  $\mathcal{M}$ ). Any subgraph of the full preemption graph of  $\delta$  constituted by all nodes representing machines sharing some job  $J$  with all edges labelled by  $J$ , is a complete graph. The corresponding subgraph in the reduced preemption graph  $G(\delta)$  is a simple path in  $G(\delta)$ . Note that to different enumerations in  $\mathcal{M}$  different preemption graphs correspond. However,  $G(\delta)$  is acyclic if and only if the preemption graph, corresponding to any enumeration in  $\mathcal{M}$  is acyclic (Shchepin & Vakhania [8]). Other important properties are given in next propositions which immediately follow.

**PROPOSITION 1** *The number of preemptions in a distribution  $\delta$  is equal to the number of edges in its preemption graph  $G(\delta)$ .*

**PROPOSITION 2** *The preemption graph of a distribution  $\delta$  is not connected iff the set of jobs  $\mathcal{J}$  can be partitioned into subsets  $\mathcal{J}_1 \cup \mathcal{J}_2 = \mathcal{J}$ , such that  $\delta(J_1) \cap \delta(J_2) = \emptyset$ , for each  $J_1 \in \mathcal{J}_1$  and  $J_2 \in \mathcal{J}_2$ .*

A distribution  $\delta$  is called *acyclic* if  $G(\delta)$  is acyclic. For a given distribution  $\delta$  and the corresponding assignment, an *assignment graph* is defined as a bipartite graph in which the two sets of nodes are formed by the sets of jobs  $\mathcal{J}$  and

machines  $\mathcal{M}$ , respectively, such that there is an edge  $(J, M)$  iff  $\delta(J, M) > 0$  (Lenstra et al. [5]). An *acyclic assignment* is an assignment with the acyclic assignment graph. There is an obvious similarity between preemptions and assignment graphs. The following proposition immediately follows from the definitions.

**PROPOSITION 3** *Two distributions with the same assignment have the same preemption graph. Hence, the assignment graph of any distribution is acyclic iff the preemption graph of this distribution is acyclic.*

## 2. NP-hardness of $P/pmt n(m-2)/C_{\max}$

In this section we show that  $P/pmt n(m-2)/C_{\max}$  is NP-hard. We use the reduction from the NP-complete PARTITION problem (see Garey & Johnson [1]) for the decision version of  $P/pmt n(m-2)/C_{\max}$ . In the PARTITION problem we are given a finite set of integer numbers  $C = \{c_1, c_2, \dots, c_n\}$  with  $S = \sum_{i=1}^n c_i$ .<sup>1</sup> This decision problem gives a “yes” answer iff there exists a subset of  $C$  which sums up to  $S/2$ . Given an arbitrary instance of a PARTITION, let us define our scheduling instance with  $n + 2m + 2$  jobs with the total length of  $2m + \frac{1}{2^m}$  as follows.

There are  $m$  pairs of the so-called *big* jobs denoted by  $B_i^\pm$ ,  $i = 1, \dots, m$  with  $|B_i^+| = 1 + 1/2^i$ , and  $|B_i^-| = 1 - 1/2^i$ . So the total length of all big jobs is  $2m$ .

There are two *median* jobs denoted by  $D$  and  $D'$ , with  $|D| = \frac{1}{2^m} - \frac{5}{m2^{m+2}}$  and  $|D'| = \frac{3}{m2^{m+2}}$ . So total length of the two median jobs is  $\frac{1}{2^m} - \frac{1}{m2^{m+1}}$ .

There are  $n$  *small* jobs  $C_i$  with  $|C_i| = \frac{c_i}{mS2^{m+1}}$ . So the total length of small jobs is  $\frac{1}{m2^{m+1}}$ .

This transformation is polynomial as the number of jobs is bounded by the polynomial in  $n$  and  $m$ , and all magnitudes can be represented in binary encoding in  $O(m)$  bits.

Now we prove that there exists a feasible schedule with less than  $m - 1$  preemptions and with the optimal makespan  $2 + \frac{1}{m2^m}$  iff there exists a solution to our PARTITION. In one direction, suppose  $\sum_{i=1}^k c_i = S/2$ , for some  $k < n$ , i.e. we have a solution to the PARTITION. Then we define a tight schedule  $\sigma$  with the makespan  $2 + \frac{1}{m2^m}$  as follows. The job sequence on  $M_1$  is:  $B_1^-, B_1^+, D', C_1, \dots, C_k$ ; so the completion time of  $M_1$  is  $2 + \frac{3}{m2^{m+2}} + \frac{1}{m2^{m+2}} = 2 + \frac{1}{m2^m}$  (the load of  $M_1$ ). The job sequence on  $M_2$

<sup>1</sup>In this and the following section  $n$  stands exclusively for the number of elements in PARTITION; in the rest of the paper  $n$  denotes the number of jobs

is:  $B_2^-, D, C_{k+1}, \dots, C_n, B_2^+$ , where we have only a part of  $D$  with the length  $\frac{3}{m2^{m+2}} = \frac{3}{4} \frac{1}{m2^m}$  (providing that the load of  $M_2$  is  $2 + \frac{1}{m2^m}$ ). The rest of  $D$  is divided into equal parts of the length  $\frac{1}{m2^m}$  and is distributed on the machines  $M_i, i > 2$ . The schedule on  $M_i, i > 2$  is tight and is generated by the sequence  $B^-, D, B^+$ .

To see that  $\sigma$  is feasible, we need to check the sequentiality of  $\sigma$  on  $D$ , which is the only preempted job in  $\sigma$ . The completion time of  $D$  on  $M_i$  is no more than  $1 - \frac{1}{2^i} + \frac{1}{m2^m} \leq 1 - \frac{1}{2^i} + \frac{1}{m2^{m+1}} \leq 1 - \frac{1}{2^i} + \frac{1}{2^{i+1}} = 1 - \frac{1}{2^{i+1}}$ , and its starting time on  $M_{i+1}$  is  $1 - \frac{1}{2^{i+1}}$ . Hence,  $\sigma$  is sequential and has the makespan  $2 + \frac{1}{m2^m}$ . This completes the proof in one direction. We need the following lemma for the other direction:

**LEMMA 1** *If there exists a uniform distribution  $\delta$  of jobs of  $\mathcal{J}$  on  $m$  identical processors from  $\mathcal{M}$  with less than  $m - 1$  preemptions, then there is a subset  $\mathcal{J}'$  of  $\mathcal{J}$  with the total length of  $k|\delta|_{\max}$ , for some natural  $k < m$ .*

**Proof.** Since  $pr(\delta) < m - 1$ ,  $G(\delta)$  has less than  $m - 1$  edges and hence any its connected component contains  $k < m$  machines (we may have two or more such components). Let  $\mathcal{J}'$  be the set of all jobs distributed on machines in any of the connected components. Since  $\delta$  is uniform, the total length of these jobs is  $k|\delta|_{\max}$ .  $\square$

Assume now  $\sigma$  is a tight  $(m - 2)$ -preemptive schedule for our scheduling instance. A distribution, generated by  $\sigma$  also has no more than  $m - 2$  preemptions and the load on all machines in this distribution is  $2 + \frac{1}{m2^m}$ . We will prove that this distribution already gives a solution to our instance of PARTITION.

Suppose  $\mathcal{J}'$  is a subset of  $\mathcal{J}$  with the total length of  $|\mathcal{J}'| = 2k + k\frac{1}{m2^m}$  (see Lemma 1). First we note that  $\mathcal{J}'$  contains exactly  $2k$  big jobs. Indeed,  $\mathcal{J}'$  cannot contain more than  $2k$  big jobs, because the total length of the smallest  $2k + 1$  big jobs is  $2k + 1 - \sum_{i=1}^{k+1} \frac{1}{2^i} = 2k + \frac{1}{2^{k+1}} \geq 2k + \frac{1}{2^m} > 2k + k\frac{1}{m2^m} = |\mathcal{J}'|$ .

At the same time, the total length of the longest  $2k - 1$  big jobs together with all median and small jobs is less than  $2k$ . Further, the total length  $B'$  of all big jobs from  $\mathcal{J}'$  is  $2k$ . Indeed if  $B' - 2k$  is not 0, it must have an absolute value of at least  $\frac{1}{2^m}$ . If  $B' < 2k$ , then  $B'$  plus the total length of all non-big jobs (which is no more than  $\frac{1}{2^m}$ ) will be no more than  $2k$ . This contradicts our assumption that  $|\mathcal{J}'| = 2k + k\frac{1}{m2^m}$ . Similarly, if  $B' > 2k$ , then the above magnitude is at least  $2k + \frac{1}{2^m} > 2k + k\frac{1}{m2^m} = |\mathcal{J}'|$ , which again is a contradiction. Thus,  $B' = 2k$ .

It follows that the total length of the big jobs from  $\mathcal{J}^c$  is  $2(m - k)$ ,  $\mathcal{J}^c$  being the complement of  $\mathcal{J}'$  in  $\mathcal{J}$ , and hence without loss of generality, we can assume that  $k \leq m/2$ . In this case,  $D$  cannot belong to  $\mathcal{J}'$ , because  $|D| > k\frac{1}{m2^m}$ . On the other hand,  $D'$  must belong to  $\mathcal{J}'$ . Indeed, denote by  $C'$

the total length of all small jobs from  $\mathcal{J}'$ . If  $\mathcal{J}'$  does not contain a median job, then  $|\mathcal{J}'| = B' + C' \leq 2k + \frac{1}{m2^{m+1}}$ , which contradicts our conjecture that  $|\mathcal{J}'| = 2k + k\frac{1}{m2^m}$ . Therefore,  $D' \in \mathcal{J}'$ . In this case  $|\mathcal{J}'| = B' + C' + |D'|$ . This implies that  $k\frac{1}{m2^m} = C' + \frac{3}{m2^{m+2}}$ . But since  $C' \leq \frac{1}{m2^{m+1}}$ , the only possible value of  $k$  is 1. Then  $C' = \frac{1}{m2^m} - \frac{3}{m2^{m+2}} = \frac{1}{m2^{m+2}}$ , which is  $S/2$  and we have obtained a solution to the PARTITION. We have proved this section's main result:

**THEOREM 2**  $P/pmtn(m-2)/C_{\max}$  is NP-hard.

### 3. The NP-hardness of $O/acyclic, pmtn(m-3)/C_{\max}$

An open shop  $\mathcal{J}, \mathcal{M}$  consists of a set of  $n$  jobs  $\mathcal{J}$  and a set of  $m$  machines  $\mathcal{M}$ . Each job  $J^j$  consists of  $m$  operations  $\{J_i^j\}$ ,  $i = 1, \dots, m$ . Each  $J_i^j$  has to be processed by machine  $M_i$  during  $|J_i^j| \geq 0$  time units, the latter is called the *length* of  $J_i^j$ . Note that we allow  $|J_i^j| = 0$ ; in this case we will say that  $J_i^j$  is a *dummy operation*.

$|J^j| = \sum_{i=1}^m |J_i^j|$  is called the *length of job  $J^j$* .  $|M_i| = \sum_{j=1}^n |J_i^j|$  is the *load* of machine  $M_i$ .

It is known that the optimal non-preemptive schedule makespan for  $O/pmtn/C_{\max}$  is the maximum between the maximal machine load and the maximal job length (see Gonzalez & Sahni [2] and Lawler and Labetoulle [4]).

A distribution  $\delta$  on  $\mathcal{J}, \mathcal{M}$  defined as  $\delta(J^j, M_i) = \frac{|J_i^j|}{|J^j|}$  can obviously be associated with each open-shop. An *acyclic open-shop* is one with an acyclic distribution. A *uniform* open-shop is defined similarly as a uniform distribution.

In this section we prove the following theorem.

**THEOREM 3**  $O/acyclic, pmtn(m-3)/C_{\max}$  is NP-hard.

The reduction from PARTITION is described in the next subsection.

#### 3.1 The reduction from PARTITION

We transform the PARTITION problem from Section 3 to  $O/pmtn/C_{\max}$ . Let  $C = \{c_1, c_2, \dots, c_n\}$  and  $S/2$ ,  $S = \sum_{i=1}^n c_i$ , form again an arbitrary instance of a PARTITION. We define our open shop instance  $O(C, m)$  as follows (assuming without loss of generality that  $m \geq 3$ ).  $O(C, m)$  deals with  $1 + n(m-2) + 2m - 2 = (n+2)m - 2n - 1$  jobs and  $m$  machines  $\{M_i\}_{i=1}^m$ . We divide the set of jobs into the following three categories:

1) There is one *common* job  $I$  to be distributed on all machines. The processing requirement of  $I$  on  $M_i$ ,  $i = 1, 2, \dots, m$ , is 1. Hence, the total processing time of  $I$  is  $m$ .

2) The jobs from the second category are called *partition jobs*. We introduce  $n$  partition jobs for each of the machines, except the first one  $M_1$  and the last one  $M_m$  (we call these machines *extremal*, and the rest of machines *intermediate*). The  $j$ th,  $j \leq n$ , partition job to be distributed on machine  $M_i$  ( $1 < i < m$ ) is denoted by  $P_{i,j}$ . The processing time of  $P_{i,j}$  is  $\frac{c_j}{S^{2i}}$ . Note that the total processing time of all partition jobs on each  $M_i$  ( $1 < i < m$ ) is equal to  $\frac{1}{S^{2i-1}}$ .

3) The jobs from the third category are called *fixers* and denoted by  $F_1, F_2^+, F_2^-, F_3^+, F_3^-, \dots, F_{m-1}^+, F_{m-1}^-, F_m$ . As it is evident from this notation, there is only one fixer on machines  $M_1$  and  $M_m$ , and there are two fixers on any intermediate machine. The processing time of  $F_1$  and  $F_m$  is  $m - 1$ . The processing time of the first fixer for machine  $M_i$ ,  $F_i^+$ ,  $1 < i < m$ , is equal to  $(i - 1) - \frac{1}{2S^{2i-1}}$ , and the processing time of the second fixer  $F_i^-$  is  $(m - i) - \frac{1}{2S^{2i-1}}$ .

Similarly as in Section 3, our transformation is polynomial. Observe that all machine loads are  $m$  and that the operations which we have not explicitly defined are dummy ones providing that  $O(C, m)$  is acyclic. The following fact is also obvious:

LEMMA 4 *Any tight schedule for  $O(C, m)$  with the makespan  $m$  is optimal.*

Given a solution  $\sum_{i \leq k} c_i = S/2$  to our PARTITION instance (we renumber elements in PARTITION correspondingly), a tight schedule for  $O(C, m)$  without any splitting and with the optimal makespan  $m$  (Lemma 4) can easily be built. On  $M_1$  first is scheduled the fixer  $F_1$  and then the common job  $I$ . On the last machine, first  $I$  is scheduled and then  $F_m$ . On each intermediate machine  $M_i$ , jobs are scheduled in the following order: first the fixer  $F_i^+$ , then all  $P_{i,j}$ ,  $j \leq k$  (in any order), then job  $I$  followed by the rest of the partition jobs  $P_{i,j}$ ,  $j > k$ , and finally the second fixer  $F_i^-$ .

It will take essentially longer to show that a feasible schedule to our open-shop yields a solution to PARTITION. We need a number of auxiliary notions and lemmas. Let us say that  $\sigma$  is a *partitioning schedule* if there is at least one intermediate machine  $M_i$  in  $\sigma$  and a time moment  $t$ , such that no partition job is executed at time  $t$  and the total processing time of the partition jobs, completed by time  $t$  is equal to the total processing time of the partition jobs, which are started at or after time  $t$ . We shall refer to  $t$  as a *partitioning time* in  $\sigma$ .

LEMMA 5 *Every feasible schedule for  $O(C, m)$  having the makespan  $m$  and at most  $m - 3$  splittings is a partitioning schedule.*



Assume for now that the above lemma is true and that  $\sigma$  is a feasible schedule for  $O(C, m)$ , such that  $\|\sigma\| = m$  and  $sp(\sigma) \leq m - 3$ . By Lemma 5,  $\sigma$  contains an intermediate machine with a switching point which is a partitioning time. Such a switching point can be found in linear time by checking all switching points, corresponding to the partition jobs on each intermediate machine. Clearly, any partitioning time gives a solution to the PARTITION problem. Thus Theorem 3 is proved under the assumption that Lemma 5 is true. The rest of this section is devoted to the proof of this lemma.

### 3.2 Schedule editing

In this subsection we introduce schedule editing operations which we use later on. It will be useful to note that if  $\sigma$  is an optimal schedule for  $O(C, m)$ , then the common job  $I$  has to be processed at any time moment in  $\sigma$  (because the processing time of  $I$  equals to the makespan of  $\sigma$ ). Next, let us note that the number of switching points is closely related with the number of splittings. The following lemma gives an explicit formula which proof is straightforward:

**LEMMA 6** *The number of splittings in a tight schedule on a machine  $M$  is equal to  $s - n - 1$ , where  $s$  is the number of all switching points on  $M$  and  $n$  is the number of jobs scheduled on  $M$ .*

The schedule editing operations are cutting, inserting and moving and are defined as follows.

**Cutting.** Let  $p$  and  $q$ ,  $p < q$ , be time moments. A schedule  $\sigma'$  is obtained from a schedule  $\sigma$  by the  $(p, q)$ -cutting on a machine  $M_i$  if the following conditions hold:

- 1  $\sigma'(J, M) = \sigma(J, M)$  for all  $M \neq M_i$  and all  $J$ ;
- 2  $\sigma'(M_i, t) = \sigma(M_i, t)$  for  $t < p$ ;
- 3  $\sigma'(M_i, t) = \sigma(M_i, t + q - p)$  for  $t \geq p$ .

Note that if  $\sigma$  is tight, then  $\sigma'$ , obtained by the  $(p, q)$ -cutting on  $M_i$ , is also tight and the completion time of  $M_i$  in  $\sigma'$  is  $p - q$  less than that in  $\sigma$ . Further, if  $p$  and  $q$  are switching points in  $\sigma$ , then the number of splittings in  $\sigma'$  cannot increase. Moreover, this number will be decreased by 1 if  $M_i$  performed the same job immediately before time  $p$  and immediately after time  $q$  in  $\sigma$ .

**Inserting.** A schedule  $\sigma'$  is obtained from a schedule  $\sigma$  by the  $(p, q)$ -inserting of a job  $J$  on machine  $M_i$  if the following conditions hold:

- 1  $\sigma'(J, M) = \sigma(J, M)$  for all  $M \neq M_i$  and all  $J$ ;
- 2  $\sigma'(M_i, t) = \sigma(M_i, t)$  for  $t \leq p$ ;

3  $\sigma'(M_i, t) = J$  for  $t \in [p, q)$ ;

4  $\sigma'(M_i, t) = \sigma(M_i, t - (q - p))$  for  $t \geq q$ .

Again,  $\sigma'$  is tight if  $\sigma$  is tight and the completion time of  $M_i$  in  $\sigma'$  is  $q - p$  greater than that in  $\sigma$ . Further, if  $p$  is a switching point in  $\sigma$  on  $M_i$ , then the inserting cannot create a splitting of any job, different from the inserted job  $J$ ; it yields a new splitting of job  $J$  if  $J$  in  $\sigma$  was already scheduled on  $M_i$  and  $p$  is not a boundary of  $\sigma(J, M_i)$ .

**Moving.** We will use  $(p, i, q)$  for the schedule component of a job  $J$  on the machine  $M_i$  with the time interval  $[p, q)$ . The  $(p, q) \rightarrow (p', q')$ -moving cuts a whole  $J$ -component  $(p, i, q)$  of  $\sigma$  from the interval  $[p, q)$  and inserts it into the interval  $[p', q')$  (of the same length) on the same machine  $M_i$ . Thus the moving is performed into two steps. The first step is the  $(p, q)$ -cutting on  $M_i$ ; with the second step, in the obtained (after cutting) schedule the  $(p', q')$ -inserting of  $J$  on  $M_i$  is carried out.

The moving does not affect the tightness of  $\sigma$  and the completion time of machine  $M_i$  if  $q' \leq |\sigma|_{M_i}$ . Note that if  $p > p'$  and  $p'$  is a switching point in  $\sigma$ , then the  $(p', q')$ -inserting will not create a new splitting of any job, scheduled on  $M_i$ . Consequently, neither the  $(p, q) \rightarrow (p', q')$ -moving of a  $J$ -component will cause a new splitting. Moreover, if  $M_i$  processed the same job immediately before time  $p$  and immediately after time  $q$  in  $\sigma$ , then the number of splittings in the resulted schedule  $\sigma'$  will be one less than that in  $\sigma$ . Now in general, if  $p$  is not a switching point, then the moving of a  $J$ -component can increase the number of splittings at most by 1. If  $M_i$  is an extremal machine, then the moving  $(p, q) \rightarrow (p', q')$  of a  $I$ -component does not increase the number of splittings if  $0 < p < q < |\sigma|_{M_i}$ . Indeed, since there are only two jobs scheduled on  $M_i$  (the common job and the fixer), the cutting of the  $I$ -component will reduce by 1 the number of splittings of the fixer.

The following lemma summarizes the above made observations.

**LEMMA 7** *Let  $\sigma'$  be a schedule produced from  $\sigma$  by the  $(p, q) \rightarrow (p', q')$ -moving of a  $J$ -component on machine  $M_i$ , such that  $q' \leq |\sigma|_{M_i}$ . Then*

1  $\sigma'$  is tight if  $\sigma$  is tight;

2  $sp(\sigma') \leq sp(\sigma) + 1$ ;

3  $sp(\sigma') \leq sp(\sigma)$  if  $p' < p$  and  $p'$  is switching point for  $\sigma$  on  $M_i$ ;

4  $sp(\sigma') < sp(\sigma)$  and  $sp_J(\sigma') < sp_J(\sigma)$  if  $p' < p$  and  $p'$  is  $J$ -switching point in  $\sigma$  on  $M_i$ ;

5  $sp(\sigma') \leq sp(\sigma)$  if  $i = 1$  or  $i = m$ .

LEMMA 8 *A feasible tight schedule for  $O(C, m)$  with no more than  $m - 2$  splittings can be constructed in linear time.*

Proof. First construct a tight schedule for all jobs except the common job  $I$ . Then for each  $M_i$ , insert  $I$  in this schedule by the  $(i - 1, i)$ -insertion. This insertion may create at most one splitting on an intermediate machine.  $\square$

### 3.3 Schedule preprocessing

In this section we introduce procedures based on cutting, inserting and moving, which simplify a schedule structure. A sequence of  $J$ -components  $(p_1, i_1, q_1), (p_2, i_2, q_2), \dots, (p_k, i_k, q_k)$  is called a  $J$ -loop, if  $q_j = p_{j+1}$  for all  $j < k$ , and  $i_1 = i_k$ ; we will say that the above  $J$ -loop passes through machines  $M_{i_2}, M_{i_3}, \dots, M_{i_{k-1}}$ .

A  $J$ -component  $(p, i, q)$  is called *unit* if  $q - p = 1$ , otherwise it is called *non-unit*. A  $J$ -loop is called *integral* if any its component, except the first one and the last one are unit, and all machines which this loop passes (the intermediate ones) have at least one splitting.

We will say that a  $J$ -component  $(p, i, q)$  splits a job  $J'$ , if both  $p$  and  $q$  are switching points of  $J'$ , i.e., if  $M_i$  processes  $J'$  immediately before time  $p$  and immediately after time  $q$ .

LEMMA 9 *If a  $J$ -component splits another job, then any its moving will not increase the total number of splittings.*

Proof. The cutting of such a  $J$ -component will decrease the number of splittings, and the insertion can again increase the number of splittings at most by 1. Hence, the total number of splittings will not increase.  $\square$

LEMMA 10 *Given any tight schedule  $\sigma$ , it is possible to construct in linear time another tight schedule  $\sigma'$  with  $sp(\sigma') \leq sp(\sigma)$ , such that any machine  $M_i$  without an  $I$ -splitting in  $\sigma$  with  $sp^\sigma(M_i) > 0$  will have at most one splitting in  $\sigma'$  (and this splitting will be caused by the unique  $I$ -component).*

Proof. Let  $(p, i, p + 1)$  be the unique  $I$ -component on  $M_i$  and let  $sp^\sigma(M_i) > 0$ . A tight schedule  $\sigma'$  with at most 1 splitting on  $M_i$  can be obtained as follows. First construct a tight schedule on  $M_i$  including in it all jobs, scheduled in  $\sigma$  on  $M_i$ , but job  $I$ . This schedule has no splittings. Then carry out the  $(p, p + 1)$  insertion of  $I$  on  $M_i$ . The resulted schedule satisfies our claim for  $M_i$ . We carry out the analogous procedure on any other machine without a  $I$ -splitting and obtain the resulted schedule  $\sigma'$  with  $sp(\sigma') \leq sp(\sigma)$ .  $\square$

LEMMA 11 *Given any tight schedule  $\sigma$ , it is possible to construct in linear time another tight schedule  $\sigma'$  without any integral loop and with  $sp(\sigma') \leq sp(\sigma)$ .*

Proof. Due to the previous Lemma 10, without loss of generality, we can assume that in  $\sigma$  all machines with unit  $I$ -components have at most 1 splitting generated by the corresponding unique  $I$ -component. If  $\sigma$  contains an integral loop  $(p, i, q), (q, i_1, q+1), (q+1, i_2, q+2), \dots, (q+k-1, i_k, q+k), (q+k, i, r)$ , then the machines  $M_{i_1}, \dots, M_{i_k}$  have just one splitting generated by the corresponding  $I$ -component. Then by Lemma 9, a moving of these components will not increase the number of splittings. Hence, the schedule  $\sigma'$ , obtained from  $\sigma$  by the following sequence of movings will have less splittings than  $\sigma$  has: on  $M_i$  perform the moving  $(q+k, r) \rightarrow (q, r-k)$ ; on  $M_{i_1}$  perform the moving  $(q, q+1) \rightarrow (r-k, r-k+1)$ ; on  $M_{i_2}$  perform the moving  $(q+1, q+2) \rightarrow (r-k+1, r-k+2)$ ; and so on, the last moving  $(q+k-1, q+k) \rightarrow (r-1, r)$  is performed on  $M_{i_k}$ . The first from the above movings decreases the number of splittings on  $M_i$  and any successive moving does not increase the number of splittings. Now if  $\sigma'$  still contains an integral loop, then we apply the same procedure again. The process has to terminate in  $sp(\sigma)$  steps.  $\square$

Let us say that an intermediate machine  $M_i$  in a schedule  $\sigma$  is *incorrect*, if  $\sigma$  has no splitting on  $M_i$  and the common job  $I$  is scheduled the first or the last on  $M_i$ .

LEMMA 12 *Given a tight schedule  $\sigma$ , associated with  $O(C, m)$ , it is possible to construct in linear time another tight schedule  $\sigma'$  with  $sp(\sigma') \leq sp(\sigma)$  and without any incorrect machine.*

Proof. Note that  $\sigma$  can have no more than two incorrect machines: on one of these machines  $I$  is scheduled first and on the other one  $I$  is scheduled last. Let  $M_i$  be the incorrect machine on which  $I$  is scheduled first (the other case will be quite similar). The schedule  $\sigma'$  which we construct from  $\sigma$  coincides with  $\sigma$  on all machines except the machines  $M_1$  and  $M_i$ . We define  $\sigma'$  on  $M_1$  as follows:  $\sigma'(I, M_1) = [0, 1)$  and  $\sigma'(F_1, M_1) = [1, m)$ .

Now let  $(p_1, 1, q_1), (p_2, 1, q_2), \dots, (p_k, 1, q_k)$  be the increasing sequence of all  $I$ -components on  $M_1$  in  $\sigma$ . As  $p_1 > 0$ , the number of splittings in  $\sigma$  on  $M_1$  is  $2k - 2$  if  $q_k = m$  and it is  $2k - 1$  otherwise.  $\sigma'$  on  $M_i$  is defined as follows. First we perform the  $[0, 1)$ -cutting on  $M_i$  which leaves 0 splittings on  $M_i$ . Further, step by step we perform  $(p_i, q_i)$ -insertions of  $I$  on  $M_i$ . Any such insertion but the last one increases the number of splittings at most by 2. The last insertion increases the number of splittings by one if  $q_k = m$ . Hence, the number of splittings in  $\sigma'$  on  $M_i$  is no more than the number of splitting in  $\sigma$  on  $M_1$ , and  $sp^\sigma(M_i) = sp^{\sigma'}(M_0) = 0$ . Therefore,  $sp(\sigma') \leq sp(\sigma)$ .  $\square$

### 3.4 Proper times

For a real  $x$ , we denote by  $\langle x \rangle$  its *fractionation* defined as the distance from  $x$  to the nearest integer number.

A time moment  $t$  is called a *proper time* for a machine  $M_i$  in a schedule  $\sigma$  if it is an integer number plus the sum of the processing times of a proper non-empty subset of the jobs, scheduled in  $\sigma$  on  $M_i$  and different from  $I$ . For example, for machines  $M_1$  and  $M_m$  any integral number is a proper time. For an intermediate machine  $M_i$  the following inequalities hold.

LEMMA 13 *The fractionation of a proper time  $t$  of an intermediate machine  $M_i$  satisfies the inequalities*

$$\frac{1}{S^{2i}} \leq \langle t \rangle \leq \frac{1}{S^{2i-1}}$$

Proof.  $t$  has a form  $f + p$ , where  $f$  is the sum of processing times of 0,1 or 2 fixers and  $p$  is the sum of processing times of a set of partition jobs. (Note that by the definition,  $f + p > 0$  and since in this sum not all fixers and partition jobs can be presented, it is not integral.) Since in  $f + p$  at least one job processing time is included, it follows immediately from the definition of processing times of partition jobs and fixers that  $\langle f + p \rangle \geq \frac{1}{S^{2i}}$ . To see the other inequality, note that  $0 \leq p \leq \frac{1}{S^{2i-1}}$  and the list of possible values of  $f$  is  $0, i - 1 - \frac{1}{2S^{2i-1}}, m - i - \frac{1}{2S^{2i-1}}$  and  $m - 1 - \frac{1}{S^{2i-1}}$ . It is straightforward to check that this implies  $\langle f + p \rangle \leq \frac{1}{S^{2i-1}}$ .  $\square$

For a machine  $M_i$ , a time moment which is not proper is called *improper*. A proper time for some machine is said to be *strongly improper*, for any other machine in  $\sigma$ .

LEMMA 14 *A strongly improper time for any intermediate machine is improper for that machine.*

Proof. The fractionation of a proper time for an intermediate machine  $M_i$  is from the interval  $[\frac{1}{S^{2i}}, \frac{1}{S^{2i-1}}]$  (Lemma 13). If  $t$  is a proper time for an intermediate machine  $M_j$ ,  $j \neq i$ , then  $\langle t \rangle$  is from the interval  $[\frac{1}{S^{2j}}, \frac{1}{S^{2j-1}}]$ . But the intersection of the two above intervals is empty. The same is true if  $M_j$  is an extremal machine, since a proper time of such a machine is integral.  $\square$

LEMMA 15 *If  $t_1, t_2$  and  $t_3$  are proper times for different machines, then  $t_1 \neq t_2 \pm t_3$ .*

Proof. Let  $t_1, t_2$  and  $t_3$  be proper times for machines  $M_i, M_j$  and  $M_k$ , respectively. Without any loss of generality, assume  $i < j < k$ . If  $i = 1$  then  $t_1$  is integer and  $t_3 \pm t_1$  are proper times for  $M_k$  and hence strongly improper for  $M_j$ . Then by Lemma 14  $t_2 \neq t_3 \pm t_1$ . The proof is similar for  $k = m$ .

Let us assume now that  $1 < i < j < k < m$ , and by a contradiction, suppose  $t_1 = t_2 + t_3$ . Then since  $S^{2j}t_1$  and  $S^{2j}t_2$  are integral,  $S^j t_3$  also has to be integral. But the inequalities  $\frac{1}{S^{2k}} \leq \langle t_3 \rangle \leq \frac{1}{S^{2k+1}}$  imply

$\frac{1}{S^{2k-2j}} \leq \langle S^{2j}t_3 \rangle \leq \frac{1}{S^{2k-2j+1}}$ . Then  $S^{2j}t_3$  cannot be integral and we came to a contradiction. The case  $t_1 = t_2 - t_3$  is similar.  $\square$

Our this sections last lemma easily follows from the definition of proper times.

LEMMA 16 *Suppose  $\sigma$  is a tight non-partitioning schedule and  $M_i$  is any correct machine in  $\sigma$  without a splitting. Then any switching point in  $M_i$  is proper.*  $\square$

### 3.5 The proof of Lemma 5

From now on we will be dealing exclusively with an associated with  $O(C, m)$  tight non-partitioning schedule without any integral loop and any incorrect machine. We denote such a schedule by  $\sigma$ . By virtue of Lemmas 11 and 12 it is sufficient to prove Lemma 5 for  $\sigma$ . We wish to prove that  $\sigma$  has at least  $m - 2$  splittings. If  $\sigma$  has a splitting on all intermediate machines, then the number of splitting is at least  $m - 2$  and our assertion is true. Hence, it remains to consider the case when the set of machines without a splitting contains an intermediate machine in  $\sigma$ .

We denote by  $s_i$  and  $f_i$ , respectively, the starting time and finishing time, respectively, of the common job  $I$  on machine  $M_i$ . Note that  $I$  is not split on  $M_i$  iff  $f_i - s_i = 1$ . Let  $M_i$  and  $M_j$  be a pair of machines without any splitting in  $\sigma$  such that  $f_i \leq s_j$ . Then we call such a pair a  $\sigma$ -successive pair in  $\sigma$  if there is no other machine  $M_k$  without splittings, such that  $f_i \leq s_k \leq f_k \leq s_j$ . Notice that since  $\sigma$  is a non-partitioning schedule and there is no splitting on machines  $M_i$  and  $M_j$ ,  $s_j$  is in fact strongly more than  $f_i$ . This implies that there must exist at least one machine with a splitting, on which a  $I$ -component is scheduled within the interval  $[f_i, s_j]$  (since job  $I$  has to be in process at any moment in  $\sigma$ ). Also observe that either  $M_i$  or  $M_j$  must be intermediate.

We will say that a  $I$ -component  $(p, k, q)$  precedes another  $I$ -component  $(p', k', q')$ , if  $q \leq p'$ ; a  $I$ -component  $(p, k, q)$  is a  $(i, j)$ -intermediate, if  $f_i \leq p \leq q \leq s_j$ . A  $I$ -component  $(p, k, q)$  is improper for an extremal machine  $M_k$  if  $p - q < 1$  and  $0 < p < q < m$ . This component is improper for an intermediate machine  $M_k$  if, besides the above two conditions for extremal machines, both  $p$  and  $q$  are improper times for  $M_k$ . An improper  $I$ -component which has one (two, respectively) strongly improper end(s) is called semi-strongly improper (strongly improper, respectively).

LEMMA 17 *Let  $M_i, M_j$  be a  $\sigma$ -successive pair. Then*

- (1) *the set of  $(i, j)$ -intermediate improper  $I$ -components is not empty;*
- (2) *if the earliest scheduled  $(i, j)$ -intermediate improper  $I$ -component belongs to an intermediate machine, then it has a strongly improper starting time;*
- (3) *if the latest scheduled  $(i, j)$ -intermediate improper  $I$ -component belongs to an intermediate machine, then it has a strongly improper finishing time.*

Proof. The difference  $s_j - f_i$  is not integral since otherwise both,  $s_j$  and  $f_i$  would be proper for both  $M_i$  and  $M_j$  and this cannot be true because at least one of these machines is intermediate (see the definition of a proper time and Lemma 14). Since the length of  $I$  coincides with the makespan of our schedule, this job is processed during the whole interval  $[s_j, f_i)$ , i.e., there is an  $I$ -component inside this interval. Let  $(p_1, k_1, q_1)$  be the earliest non-unit  $I$ -component after  $f_i$ . If  $M_{k_1}$  is an extremal machine, then this  $I$  component is improper (since it is non-unit) and (1) is proved. If  $M_{k_1}$  is an intermediate machine, then  $p_1$  may differ from  $f_i$  only by an integer (the sum of processing times of unit  $I$ -components), hence it is proper for  $M_i$  and strongly improper for  $M_{k_1}$ . If  $q_1$  is improper for  $M_{k_1}$ , then this component is improper and statements (1) and (2) are proved. Assume  $q_1$  is proper for  $M_{k_1}$ . Then we continue to search an improper  $I$ -component. The difference  $s_j - q_1$  is again non-integral as it is a difference of proper times of different machines. Hence, there is at least one non-unit  $I$ -component scheduled between  $q_1$  and  $s_j$ . Let  $(p_2, k_2, q_2)$  be the earliest such component. If  $k_2 = 1$  or  $k_2 = m$ , this component is improper and the lemma is proved. Assume  $1 < k_2 < m$ . Because  $\sigma$  has no integral loops,  $k_2 \neq k_1$ . Since  $p_2$  is the sum of  $f_1$  and an integer number, it is proper for  $M_{k_1}$  and hence strongly improper for  $M_{k_2}$ . If  $q_2$  is improper,  $(p_2, k_2, q_2)$  is improper and the statements (1) and (2) are proved. If not, we continue the procedure and look for the next  $I$ -component  $(p_3, k_3, q_3)$  and so on. This process cannot take more steps than the number of splittings of  $\sigma$ , and we obtain an improper  $I$ -component with strongly improper start time. The statements (1) and (2) are proved.

To prove (3), we merely replace in the above the "first" by the "last".  $\square$

We use the above lemma to mark intermediate improper  $I$ -components for every  $\sigma$ -successive pair  $(M_i, M_j)$ . In particular, if there is an  $(i, j)$ -intermediate improper  $I$ -component on an extremal machine, then we mark the earliest such component. Otherwise, if there is only one  $(i, j)$ -intermediate improper  $I$ -component, we mark this piece (such component will be strongly improper by Lemma 17). If there is more than one  $(i, j)$ -intermediate improper  $I$ -component, we mark the earliest and the latest of these  $I$ -components; we call a couple of such components a *twin couple*.

Let us denote by  $mr(M)$  the number of the marked  $I$ -components, scheduled on machine  $M$ ; if  $mr(M) > 0$ , then we call  $M$  *marked*. Let  $M$  be a machine with at least one splitting. Then the number of *spare splittings* on a  $M$ ,  $sp^s(M) = sp(M) - 1$ .

LEMMA 18 *If  $M$  is an extremal marked machine, then  $sp^s(M) \geq mr(M)$ .*

Proof. Note that the number of switching points on  $M$  is at least  $2mr(M) + 2$ . Using Lemma 6 with  $n = 2$  and  $s = 2mr(M) + 2$ , we obtain that  $sp(M) \geq 2mr(M) + 2 - 2 - 1 = 2mr(M) - 1$ . If  $mr(M) > 1$ , then  $2mr(M) - 1 \geq$

$mr(M) + 1$ ; if  $mr(M) = 1$ , then job  $I$  has at least one splitting and the fixer also has at least one splitting because the marked component is scheduled in an intermediate position on  $M$  and we have found at least 1 spare splitting. This completes the proof.  $\square$

LEMMA 19 *If an intermediate machine  $M_k$  is such that  $mr(M_k) > 2$ , then  $sp^s(M_k) \geq mr(M_k)$ .*

Proof. We first consider the case when there is a non-marked  $I$ -component on  $M_k$ , or equivalently  $sp_I(M_k) \geq mr(M_k)$ . Then to prove our lemma it is sufficient to find some split job on  $M_k$ , different from job  $I$ . Let  $p_1$  be the starting time of the earliest marked  $I$ -component on  $M_k$ . There are scheduled only components of jobs, different from  $I$  in the interval  $(0, p_1)$ , and at least one of these components has to be fractional since the length of this interval is improper.

Thus we can assume that  $sp_I(M_k) < mr(M_k)$ , i.e., all  $I$ -components on  $M_k$  are marked. Consider next the case when we have three marked  $I$ -components  $(p_1, k, q_1)$ ,  $(p_2, k, q_2)$  and  $(p_3, k, q_3)$ ,  $p_1 < p_2 < p_3$  which do not contain a twin couple. Hence all these pieces are intermediate for some three different  $\sigma$ -successive pairs. In this case each of the following intervals  $(0, p_1)$ ,  $(q_1, p_2)$ ,  $(q_2, p_3)$  and  $(p_3, m)$  has the length more than 1. Therefore, in each of these intervals has to be scheduled a fixer. If the same fixer is scheduled in two different intervals, then it has a splitting; if a fixer is scheduled in three different intervals, it has at least two splitting. As it is easily seen, in all cases the total number of splittings of a fixer is no less than 2. On the other hand, the number of splittings of job  $I$  is at least  $mr(M_k) - 1$ . Hence the total number of splittings on  $M_k$  is at least  $mr(M_k) - 1 + 2 = mr(M_k) + 1$ .

Now let us consider the case when there are 3 marked components on  $M_k$ , but first two of them form a twin couple. Since  $p_1$  and  $q_3$  are improper, the intervals  $(0, p_1)$  and  $(q_3, m)$  contain a fractional component of some job, say  $J_1$  for  $(0, p_1)$  and  $J_2$  for  $(q_3, m)$ . If  $J_1 \neq J_2$ , then  $sp(M_k) \geq sp_I(M_k) + 2 \geq mr(M_k) - 1 + 2$  and our lemma holds. Suppose  $J_1 = J_2$ . Observe that from Lemma 17, both  $p_1$  and  $q_2$  are strongly improper. Besides, either  $p_3$  or  $q_3$  is also strongly improper. Assume first that  $p_3$  is strongly improper. Then the interval  $(q_2, p_3)$  has an improper length by Lemma 15 and hence contains a fractional component of some job, say  $J_3$ . If  $J_3 \neq J_1$ , the reasoning, similar to that used for  $J_2$  above, proves our lemma. If  $J_3 = J_1$ , then  $J_1$  has at least 2 splittings and so  $sp(M_k) \geq sp_I(M_k) + sp_{J_1}(M_k) \geq mr(M_k) - 1 + 2$  and the lemma is also proved.

Now assume that  $q_3$  is strongly improper. Since all  $I$ -components on  $M_k$  are marked, the sum of the length of all  $I$ -intervals  $(p_i, q_i)$  is 1, and the sum of the length of the intervals  $(q_1, p_2)$  and  $(q_2, p_3)$  is respectively  $1 + q_3 - p_1$  and it is improper by Lemma 15. Hence the length of one of these intervals is



improper by Lemma 15 and it must contain a fractional job component. The proof now is completed similarly as for the above considered case.

The case when a single marked  $I$ -component follows after a twin couple is analogous. Finally, the only non-considered case is when there are four marked  $I$ -components on  $M_k$ , which form 2 different twin couples. Then the following three intervals  $(0, p_1)$ ,  $(q_2, p_3)$  and  $(q_4, m)$  have improper length and hence contain a fractional component of a job, different from  $I$ , and we again have found an additional spare splitting.  $\square$

LEMMA 20 *If  $mr(M_k) = 2$  and at least one of the marked components on  $M_k$  is strongly improper, then  $sp^s(M_k) \geq 2$ .*

Proof. Suppose  $(p, k, q)$  is a strongly improper marked  $I$ -component. The second marked  $I$ -component  $(p', k, q')$  is then a semi-strongly improper. Assume  $q < p'$  and  $p'$  is strongly improper (and hence  $q'$  is improper). The length of the interval  $(0, p)$  is improper for  $M_k$  (Lemma 17), hence it contains a fractional component of some job, say  $J_1$ . The same is true for the interval  $(q', m)$ . If this interval contains a fractional component of some another job  $J_2$ , then we already have founded three split jobs  $I$ ,  $J_1$  and  $J_2$  and the lemma holds. Suppose  $J_2 = J_1$ . If  $I$  has more than 2 components on  $M_k$ , then  $I$  has 2 splittings and  $J_1$  gives another splitting, so we again have 3 splittings on  $M_k$ . Suppose  $I$  has just 1 splitting on  $M_k$ . In this case the length of interval  $(q, p')$  is exactly 1 less than the length of  $(p, q')$ , whereas the last interval is improper as the difference of two strongly improper times (Lemma 15). Hence, the length of  $(q, p')$  is improper and this interval contains a fractional job component. If this component is of job  $J_1$ , then  $J_1$  has at least two splittings, and we have 3 splittings on  $M_k$  together with the splitting of job  $I$ . If the above component is of some other job, we again have 3 split jobs.

The case when  $q'$  is strongly improper is simpler, because then we immediately have that the length of the interval  $(p, q')$  is improper as the difference of two strongly improper times. The case  $q' < p$  is analogous.  $\square$

LEMMA 21 *Every marked machine  $M_k$  has at least 2 splittings.*

If  $I$  has more than 1 splitting on  $M_k$ , then we have enough splittings. Assume that  $I$  has just 1 splitting and let  $(p, k, q)$  be a marked  $I$ -component. Either the interval  $(0, p)$  or the interval  $(q, m)$  does not contain a component of job  $I$ . As that interval, in addition, has an improper length, it must contain a fractional component of some job, different from  $I$ . So we have found two split jobs on  $M_k$  and the lemma is proved.  $\square$

*The proof of Lemma 5.* Now we are ready to prove the main Lemma 5. Let  $k$  and  $m - k$  be the number of machines without and with splittings, respectively. The total number of splittings is  $m - k$  plus the number of spare splittings. Hence it is sufficient to prove that  $sp^s(\sigma) \geq m - 2 - (m - k) = k - 2$ .

Denote by  $p$  the number of  $\sigma$ -successive pairs for which a twin couple is marked. Then the total number of the marked components is  $k - 1 + p$ . Indeed, the number of  $\sigma$ -successive pairs is  $k - 1$ . Each such pair generates one or two marked components, while the number of the pairs which generate two components is  $p$ .

Let us call a marked machine *distinctive*, if it contains exactly two marked components, such that each of these components is from some twin couple (the two components may belong to different twin couples). We denote by  $q$  the number of all distinctive machines. Since the total number of twins is  $2p$  and the number of twins, scheduled on a distinctive machine is  $2q$ , we have that  $2q \leq 2p$ , or equivalently  $q \leq p$ .

By Lemma 21, each distinctive machine has at least 1 spare splitting. Hence, the distinctive machines give in total at least  $q$  spare splittings. The number of the marked components, scheduled on non-distinctive machines is  $(k - 1 + p) - 2q \geq k - 1 - q$ . By virtue of Lemmas 20, 19 and 18, each marked component on a non-distinctive machine generates at list 1 spare splitting. Hence, the total number of spare splittings is at least  $q + (k - 1 - q) = k - 1$  (this estimation is even better than the claimed one; it holds only if there exists an intermediate machine without splittings).  $\square$

#### 4. NP-hardness of $R/p_{ij} \leq C_{\max}^*, pmtn(2m - 4)/C_{\max}$

In this section we apply our previous NP-hardness result for  $O/acyclic, pmtn(m - 3)/C_{\max}$  and show that  $R/p_{ij} \leq C_{\max}^*, pmtn(2m - 4)/C_{\max}$  is also NP-hard. First we prove the following auxiliary lemma:

**LEMMA 22** *Two distributions  $\delta$  and  $\delta'$  are equal if they have the same acyclic assignment and the load on all machines in both  $\delta$  and  $\delta'$  is the same.*

**Proof.** Assume  $\delta$  and  $\delta'$  are two arbitrary distributions satisfying the lemma. The proof is by the induction on the number of machines. Suppose the lemma is proved for distributions with  $< m$  machines.

Let  $\delta$  and  $\delta'$  be two acyclic distributions with  $m$  machines from  $\mathcal{M}$ . Since  $\delta$  and  $\delta'$  have the same acyclic preemption graph, there is a node of degree one  $N$  in this graph. Let  $I$  be the job, corresponding to the (only) edge of  $M$ . Note that the rest of the jobs, distributed in  $\delta$  and  $\delta'$  on  $M$  have no preemption, i.e., they are assigned completely to  $M$ . Therefore, the length of all these jobs in  $\delta$  and  $\delta'$  is the same. But since the load of  $M$  in  $\delta$  and  $\delta'$  is the same, the length of the portion of  $I$  on  $M$  in both  $\delta$  and  $\delta'$ , must be also the same. Hence  $\delta(J, M) = \delta'(J, M)$  for all  $J$ . To apply the induction hypothesis, we define distributions  $\delta_0$  and  $\delta'_0$  on  $\mathcal{M} \setminus M$  in the following way:  $\delta_0(J, M) = \delta(J, M)$  if  $J \neq I$  and  $\delta(J, M) > 0$ ,  $\delta_0(I, M) = \delta(I, M) \frac{1}{1 - \delta(I, M)}$ ;  $\delta'_0(J, M) = \delta'(J, M)$

if  $J \neq I$  and  $\delta'(J, M) > 0$ ,  $\delta'_0(I, M) = \delta'(I, M) \frac{1}{1 - \delta'(I, M)}$ . Distributions  $\delta_0$  and  $\delta'_0$  coincide by induction hypothesis. This implies that  $\delta = \delta'$ .  $\square$

Given a multiprocessor  $\mathcal{J}, \mathcal{M}$ , let us say that a distribution  $\delta$  on  $\mathcal{J}, \mathcal{M}$  generates an open shop  $O$  on  $\mathcal{J}, \mathcal{M}$ , if  $|J_i^j| = \delta(J^j, M_i)M_i(J^j)$ .

**THEOREM 23** *For any uniform acyclic open shop  $O$  on  $\mathcal{J}, \mathcal{M}$ , there is a processing time function  $f$  and a distribution  $\delta$  for the multiprocessor  $\mathcal{J}, \mathcal{M}$  with this processing function, such that  $\delta$  generates  $O$  and  $\delta$  is a unique optimal distribution for  $\mathcal{J}, \mathcal{M}$ .*

*Proof.* First we define  $f$  as follows:

$M_i(J^j) = |J^j| + \varepsilon$  if  $J_i^j$  is dummy, and  $M_i(J^j) = |J^j|$  otherwise, where  $\varepsilon$  is a positive real number. Now we define the distribution  $\delta$  as  $\delta(J^j, M_i) = |J_i^j|/|J^j|$ . Note that if  $|J_i^j| > 0$ , then  $\delta(J^j, M_i)M_i(J^j) = M_i(J^j) \frac{|J_i^j|}{|J^j|} = |J_i^j| \frac{|J_i^j|}{|J^j|} = |J_i^j|$ . Similarly,  $|J_i^j| = 0$  implies  $\delta(J^j, M_i) = 0$ . Hence,  $(M_i)_\delta(J^j) = |J_i^j|$  holds for all  $i, j$  and  $\delta$  generates  $O$ . Since  $O$  is uniform,  $\delta$  is also uniform and the total processing time of  $\delta$  is  $m|\delta|_{\max}$ . Besides, every job in  $\delta$  is distributed on its fastest machine (i.e. the machine where this job can be processed in the minimal time). This implies that  $\delta$  has the minimal possible total processing time, and since  $\delta$  is uniform, it is optimal.

It remains to show that  $\delta$  is unique. Assume  $\delta'$  is another optimal distribution such that  $\delta'$  generates  $O$ . The total processing time in  $\delta'$  is no more than  $m|\delta'|_{\max}$ , but the latter by our assumption is no more than  $m|\delta|_{\max}$ ; now since  $m|\delta|_{\max}$  is the minimal possible total processing time, the total processing time in  $\delta'$  is to be equal to  $m|\delta|_{\max}$ . Hence,  $m|\delta'|_{\max} = m|\delta|_{\max}$  and  $\delta'$  is also uniform, i.e., loads on all machines in both,  $\delta$  and  $\delta'$  are equal. At the same time,  $\delta'$  has to distribute each job on the fastest machine. But the processing time function is defined in such a way that a machine  $M$  is fastest for a job  $J$  iff  $\delta(J, M) > 0$ . Hence,  $\delta'$  and  $\delta$  have the same acyclic assignment and our claim follows from Lemma 22.  $\square$

**THEOREM 24**  $R/p_{ij} \leq C_{\max}^*, pmtn(2m - 4)/C_{\max}$  is NP-hard.

*Proof.* We prove that the corresponding decision problem is NP-complete. Recall that the open-shop problem  $O(C, m)$  of Section 4 is acyclic and uniform. We apply Theorem 23 with  $\varepsilon \leq 1$  to  $O(C, m)$  to construct a processing time function  $f$  for multiprocessor  $\mathcal{J}, \mathcal{M}$ , and a distribution  $\delta$ , such that  $\delta$  generate  $O(C, m)$ . Note that  $|\delta|_{\max} = m$  and  $p_{\max} = m$ , hence the multiprocessor  $\mathcal{J}, \mathcal{M}$ , defined by the processing time function  $f$ , is non-lazy (i.e.,  $p_{\max} \leq C_{\max}$ ).

It is not difficult to see that our decision problem, “does there exist a feasible schedule  $\sigma$  for the multiprocessor  $\mathcal{J}, \mathcal{M}$  with  $\|\sigma\| \leq m$  and with  $pr(\sigma) \leq$

$2m - 4$ ?", has a "yes" answer if and only if there exists a tight schedule for  $O(C, m)$  with the makespan, not exceeding  $m$  and with less than  $m - 2$  splittings. Indeed, if  $\sigma$  is such a schedule for  $O(C, m)$ , then  $\sigma$  is a feasible schedule for the multiprocessor  $\mathcal{J}, \mathcal{M}$  with less than  $(m - 1) + (m - 2) = 2m - 3$  preemptions.

In the other direction, suppose  $\sigma$  is a feasible schedule for  $\mathcal{J}, \mathcal{M}$  with  $\|\sigma\| \leq m$  and  $pr(\sigma) < 2m - 3$ . Then the makespan of any distribution associated with  $\sigma$  is  $m$  and it is optimal. Since there is only one such distribution (Lemma 23), it is precisely the distribution  $\delta$ .  $\delta$  has exactly  $m - 1$  preemptions. It follows that  $\sigma$  is a feasible schedule for  $O(C, m)$  with the number of splittings  $pr(\sigma) - (m - 1) < m - 2$ .  $\square$

## 5. Polynomial approximation for little-preemptive multiprocessor scheduling

As mentioned earlier in Section 1, the NP-hardness results for  $P/pmtn(m - 2)/C_{\max}$  and  $O/acyclic, pmtn(m - 3)/C_{\max}$ , respectively, are tight due to the polynomial  $(m - 1)$  and  $(m - 2)$ -preemptive algorithms for  $P/pmtn/C_{\max}$  and  $O/acyclic, pmtn(m - 2)/C_{\max}$ , respectively, from McNaughton [6] and Shchepin & Vakhania [11], respectively. The tightness of the NP-hardness result for  $R/p_{ij} \leq C_{\max}^*, pmtn(2m - 4)/C_{\max}$  follows from the following theorem. The proof of this and the next theorem are based on the earlier results from Shchepin & Vakhania [9].

**THEOREM 25**  *$R/p_{ij} \leq C_{\max}^*, pmtn(2m - 3)/C_{\max}$  is polynomially solvable.*

**Proof.** An optimal acyclic distribution  $\delta$  with  $pr(\delta) \leq m - 1$  and  $|\delta|_{\max} \leq C_{\max}^*$  can be obtained by linear programming. For each  $J \in \mathcal{J}$ , we have  $|J|_{\delta}^{\delta} = \sum_{M \in \mathcal{M}} \delta(J, M)M(J) \leq \sum_{M \in \mathcal{M}} \delta(J, M)p_{\max} = p_{\max} \leq C_{\max}^*$ . Hence  $\|\delta\| \leq C_{\max}^*$  and we can apply Lemma 4 from [9] to construct an optimal schedule with less than  $m - 2$  splittings for  $\delta$ . Since  $\delta$  has at most  $m - 1$  preemptions, the total number of preemptions of the constructed schedule will be no more than  $(m - 1) + (m - 2) = 2m - 3$ .  $\square$

For general unrelated processor system we can guarantee a near-optimal approximation as the following theorem shows:

**THEOREM 26** *There is a polynomial time algorithm, which for any instance of  $R/pmtn/C_{\max}$  constructs a feasible schedule  $\sigma$  with  $pr(\sigma) \leq 2m - 3$  and with*

- (a)  $\|\sigma\| \leq \max\{C_{\max}^*, p_{\max}\};$
- (b)  $\|\sigma\| \leq C_{\max}^0$ , where  $C_{\max}^0$  is the optimal non-preemptive schedule makespan.

Proof. Part (a) immediately follows from Theorem 25. We refer the reader to Theorem 2 from [9] for the proof of part (b).  $\square$

## 6. Further research

We have shown that  $O/acyclic, pmtn(m-3)/C_{\max}$ ,  $P/pmtn(m-2)/C_{\max}$  and  $R/p_{ij} \leq C_{\max}^*, pmtn(2m-4)/C_{\max}$  are NP-hard in the normal sense. Can there exist pseudo-polynomial algorithms for (some of) these problems or are they (some of them) strongly NP-hard? A polynomial algorithm for  $R/pmtn/C_{\max}$  by Lawler and Labetoulle [4] gives up to  $(4m^2 - 5m + 2)$  preemptions, whereas our result for  $R/p_{ij} \leq C_{\max}^*, pmtn/C_{\max}$  ensures that the critical number of preemptions for  $R/pmtn/C_{\max}$  is no less than  $(2m-4)$ . Can this lapse be reduced?

## References

- [1] Garey M.R. and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness (Freeman, San Francisco, 1979)
- [2] Gonzalez T. and S. Sahni, "Open Shop Scheduling to Minimize Finish time", *Journal of the ACM* 23, 665-679 (1976)
- [3] Gonzalez T. and S. Sahni, "Preemptive scheduling of uniform processor systems", *Journal of the ACM* 25, 92-101 (1978)
- [4] Lawler E.L. and J.Labetoulle, "On preemptive scheduling of unrelated parallel processors by linear programming", *Journal of the ACM* 25, 612-619 (1978)
- [5] Lenstra J.K., D. B. Shmoys and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines" *Mathematical Programming*, 46, 259-271 (1990)
- [6] McNaughton, "Scheduling with deadlines and loss functions", *Management Science* 6, p.1-12 (1959)
- [7] Shachnai H., T. Tamir and G. Woeginger, "Minimizing makespan and preemption costs on a system of uniform machines", *Proc. 10th European Symposium on Algorithms* (2002)
- [8] Shchepin E. and N. Vakhania, "Task distributions on multiprocessor systems", In: *Lecture Notes in Computer Science (IFIP TCS)* 1872, p.112-125 (2000)
- [9] Shchepin E. and N. Vakhania, "Little-preemptive scheduling on unrelated processors". *Journal of Math. Modelling and Algorithms* 1, p. 43-56 (2002)
- [10] Shchepin E. and N. Vakhania. "An optimal rounding gives a better approximation for scheduling unrelated machines". *Operations Research Letters* 33, p.127-133 (2005)
- [11] Shchepin E. and N. Vakhania, "On machine dependency in shop scheduling", submitted to *Discrete Applied Mathematics*

# Single-Machine Scheduling with General Costs under Compound-Type Distributions\*

Xiaoqiang CAI<sup>1</sup>, Xianyi WU<sup>2</sup>, and Xian ZHOU<sup>3</sup>

<sup>1</sup>Department of Systems Engineering and Engineering Management,  
The Chinese University of Hong Kong, Hong Kong

<sup>2</sup>Department of Statistics  
East China Normal University, Shanghai

<sup>3</sup>Department of Applied Mathematics  
The Hong Kong Polytechnic University, Hong Kong

## Abstract.

We investigate in this paper the problem of scheduling  $n$  jobs on a single machine with general stochastic cost functions, which involves the effects of stochastic due times, and with the stochastic processing times in a class of distributions, which covers the family of exponential distributions, the family of geometrical distributions and some other important distribution families as its special cases. Such a class of distributions is characterized by their characteristic functions. The optimal policies for these scheduling problems, without precedence constraints or with precedence constraints of nonpreemptive chains, are discussed respectively.

**Keywords:** Stochastic scheduling, single machine, stochastic cost functions, due dates, characteristic functions, increment order, convexity order.

## 1. Introduction

Consider a stochastic scheduling problem as follows. A set of  $n$  jobs are to be processed on a single-machine, which are all available at time zero. The processing times  $P_i$  of job  $i$ ,  $i = 1, 2, \dots, n$ , are independent random variables. The cost functions are stochastic processes which can include various deterministic and/or stochastic attributes such as due dates  $D_i$ , weights  $w_i$ , etc. The problem is to determine a job sequence  $\pi$ , a permutation of the integers  $\{1, 2, \dots, n\}$ , so as to minimize the following objective function:

$$ETC(\pi) = \mathbf{E} \left[ \sum_{i=1}^n f_i(C_i(\pi)) \right], \quad (1.1)$$

---

\* This research was partially supported by the Research Grants Council of Hong Kong under Earmarked Grants No. PolyU 5285/04E.

which represents the expected total cost with general stochastic cost functions  $f_i(\cdot)$ , where for each  $i \in \{1, 2, \dots, n\}$ ,

- (1)  $\{f_i(t) : t \geq 0\}$  is a stochastic process independent of  $P_1, \dots, P_n$ ,
- (2)  $C_i(\pi)$  is the completion time of job  $i$  under a sequence  $\pi$ , and
- (3)  $\mathbf{E}(X)$  represents the expectation of a random variable  $X$ .

In particular, we consider the objective function of the form:

$$ETC(\pi) = \mathbf{E} \left[ \sum_{i=1}^n w_i g(C_i(\pi) - D_i) \right], \quad (1.2)$$

where

- (1)  $g(\cdot)$  is a nondecreasing (deterministic) function, which is supposed to have finite mean function  $m_i(t) = \mathbf{E}[f_i(t)]$  for all  $t$ , and is nondecreasing in  $t$ ,
- (2)  $w_i, i = 1, 2, \dots, n$ , are deterministic weights, and
- (3)  $D_i, i = 1, 2, \dots, n$ , are random due dates independent of the processing times  $\{P_i\}$ .

Both (1.1) and (1.2) are general objective functions, although (1.2) is a special case of (1.1). The function  $g$  in (1.2) may take a role in some situations that is similar to one played by a utility function in econometrics. Analogous to the arguments in econometrics, a convex or concave function  $g$ , which measures the attitude towards the seriousness of lateness of jobs regarding the due dates, is of considerable importance in such situations. There have been many traditional performance measures of practical importance in the literature, such as the expected weighted sum of completion times (see Rothkopf, 1966), expected weighted sum of squared completion times (see Bagga and Kalra, 1981), expected total weighted tardiness (see, Pinedo, 1983), expected weighted number of tardy jobs (see, Pinedo, 1983 and Boxma and Forst 1986), and weighted lateness probability (see Serin, Erel and Steiner, 1991), and so on, which are special cases of (1.2). A relatively detailed review on related problems was presented by Cai and Zhou (2005).

In this paper we study the problem of single-machine scheduling under a novel class of distributions for the processing times, in which the distributions are characterized by a common form of their characteristic functions. To be specific, we consider a family of distributions with a characteristic function of the form  $\phi(t, \gamma) = (1 + G(t)/\gamma)^{-1}$ , where  $\gamma$  is a parameter used to identify the distribution in the family and  $G(t)$  is a complex-valued function of real variable  $t$  such that  $\phi(t, \gamma)$  is nonnegative definite (to ensure it is a characteristic function of certain probability distribution). More details of this distribution class will be discussed in Section 2.

Similar to the family of exponential distributions, the effects of a job interchange procedure under this new family of distributions can be easily computed such that the conditions for a sequence to minimize the cost function (1.1) or (1.2) can be readily verified. For brevity, denote the model in (1.1) by  $1|P_j \sim (1 + G(t)/\gamma_j)^{-1}|E[\sum f_j(C_j)]$  and in (1.2) by  $1|P_j \sim (1 + G(t)/\gamma_j)^{-1}|E[\sum w_j g(C_j - D_j)]$ .



Generally speaking, the processing times on jobs,  $\{P_i, 1 \leq i \leq n\}$ , are nonnegative variables. But in some circumstances, it would be convenient to approximate the distribution of a processing time with one that can take negative values. An important case is to approximate the distribution of a processing time with a normal distribution. See, for example, Boys, Glazebrook and McCrone (1997), Jang, Cerry and Klein (2002) and the references therein. So we allow the processing times to be real-valued random variables with positive means in this paper.

We will show that the non-increasing order of the increments of  $\gamma_j \mathbf{E}[f_j(t)]$  is optimal for the problem  $1|P_j \sim (1 + G(t)/\gamma_j)^{-1}|E[\sum f_j(C_j)]$ . Furthermore, if the due dates  $D_j$ ,  $j = 1, \dots, n$ , have a common distribution, then the optimal policy for problem  $1|P_j \sim (1 + G(t)/\gamma_j)^{-1}|E[\sum w_j g(C_j - D_j)]$  is to schedule the jobs according to non-increasing order of  $w_j \gamma_j$ , which is the well known WSEPT (Weighted Shortest Expected Processing Time) rule. On the other hand, if the due dates  $D_j$  have different distributions, then the optimal policy depends on the distributions and relies on convex or concave  $g$ . Specifically, non-increasing order of  $w_j \gamma_j F_j(x)$  (or  $w_j \gamma_j S_j(x)$ ) is optimal when  $g$  is convex (or concave, respectively). These will be discussed in Section 3.

Furthermore, the constraints of precedence between some jobs in the form of non-preemptive chains are discussed and the results parallel to those without precedence constraints are derived in Section 4.

## 2. Classes of Compound-type Distributions

We now introduce a class of distributions with characteristic functions of the form

$$\phi(t) = (1 + G(t)/\gamma)^{-1} \quad (2.1)$$

where  $\gamma$  is a parameter identifying the distribution. Some of the most commonly used distributions are included in this class of distributions, as shown in the following examples.

**Example 1. (Exponential)** If  $P$  follows the exponential distribution with rate  $\gamma$ , that is,  $\Pr(P \leq x) = 1 - e^{-\gamma x}$ , then the characteristic function of  $P$  is  $\phi(t) = (1 - it/\gamma)^{-1}$ , where  $i$  is the imaginary unit.

**Example 2. (Laplace)** If  $P$  has a Laplace distribution with scale parameter  $\alpha$ , whose density is of the form  $(2\alpha)^{-1}e^{-|x|/\alpha}$ , then its characteristic function is  $\phi(t) = (1 + \alpha^2 t^2)^{-1}$ .

**Example 3.** Let  $\phi(t) = (1 + C|t|^\alpha)^{-1}$ , where  $0 < \alpha \leq 2$  and  $C \geq 0$ , which has the form in (2.1) if  $\alpha$  is known. Then  $\phi(t)$  is the characteristic function of Pólya-type, see, for example, Bisgaard and Zoltan (2000).

**Example 4. (Geometric)** If  $P$  has a geometric distribution with  $\Pr(P = k) = (1 - \alpha)\alpha^k$ ,  $k = 0, 1, 2, \dots$ , where  $0 < \alpha < 1$ , then  $\phi(t) = (1 + \alpha(1 - e^{it})/(1 - \alpha))^{-1}$ , which is also of the type in (2.1).

To identify all distributions with characteristic functions in (2.1) may be difficult. But finding some important special cases is less difficult. Two of such cases are given below.

**Example 5. Compound Geometry Distribution.** Let  $\{X_n\}_{n=1}^\infty$  be a series of independent and identically distributed random variables with arbitrary common distribution function,

say  $F(x)$ ,  $N$  a variable independent of  $\{X_n\}_{n=1}^{\infty}$ . Let  $X = \sum_{n=1}^N X_n$  be the partial sum of the first  $N$  items of  $\{X_n\}$ . Then  $X$  is said to have a *compound distribution*, whose name is determined by the distribution of  $N$ . For example, if  $N$  is geometrically distributed with  $\Pr(N = n) = (1 - \theta)\theta^{n-1}$ ,  $n = 1, 2, \dots$ , denoted by  $N \sim \text{Geo}(\theta)$ , then  $X$  is said to be compound geometrically distributed. If  $N$  can take value 0, that is,  $\Pr(N = k) = (1 - \theta)\theta^k$ ,  $n = 0, 1, \dots$ , then  $N$  is said to follow standard geometry distribution and denoted by  $N \sim \text{Geo}_s(\theta)$ . In such a situation, the distribution of  $X$  is referred to as the *standard compound geometric distribution*.

The compound geometric distributions arise in some practical situations of scheduling. For example, consider the situation where the processing of a job consists of many subtasks whose processing times are independently and identically distributed. The total processing time of the job is then compound geometrically distributed if the number of subtasks has a geometric distribution.

The following proposition characterizes the compound geometric distribution by its characteristic functions.

**Proposition 2.1.** If  $X$  is compound geometrically distributed, then its characteristic function has the form of (2.1).

*Proof.* We first consider the case of standard compound geometric distribution. Let  $X = \sum_{n=1}^N X_n$  be compound geometrically distributed, where  $X_1, X_2, \dots$  are independent and identically distributed with a common characteristic function  $\phi(t)$  and  $N \sim \text{Geo}_s(\theta)$ . Then the characteristic function of  $X$  is

$$\phi_X(t) = \mathbf{E}[e^{itX}] = \mathbf{E}\left[\mathbf{E}\left[\exp\left\{it \sum_{n=1}^N X_n\right\} \middle| N\right]\right] = E(\phi^N(t)) = (1 - \theta) \sum_{n=0}^{\infty} [\theta\phi(t)]^n.$$

Since  $|\phi(t)\theta| < 1$ , the series converges and

$$\phi_X(t) = \frac{1 - \theta}{1 - \theta\phi(t)} = \left[1 + \frac{\theta}{1 - \theta}(1 - \phi(t))\right]^{-1} \quad (2.2)$$

Similarly, if  $N \sim \text{Geo}(\theta)$  is non-standard geometric, then

$$\begin{aligned} \phi_X(t) &= \mathbf{E}[e^{itX}] = \mathbf{E}\left[\mathbf{E}(\exp\{it \sum_{n=1}^N X_n\} \middle| N)\right] = E[\phi^N(t)] \\ &= (1 - \theta)\phi(t) \sum_{n=0}^{\infty} [\theta\phi(t)]^n = \frac{(1 - \theta)\phi(t)}{1 - \theta\phi(t)} = \left[1 + \frac{1}{1 - \theta} \left(\frac{1}{\phi(t)} - 1\right)\right]^{-1}. \end{aligned} \quad (2.3)$$

So the compound geometric distributions (whether standard or not) have characteristic functions of the form in (2.1). ■

**Example 6.** *Lévy Process with Exponential Distributed Horizon.*

Suppose that  $\{X(t) : t \in I\}$  is an arbitrary stochastic process with independent increments, where  $I$  is the set of its (time) horizon. Let  $T$  be a random variable taking

values in  $I$ , independent of the process  $\{X(t) : t \in I\}$ . The random variable  $X(T)$  is termed as a *process with random horizon*, and its distribution is called a *generalized compound distribution*. For simplicity, we will only discuss the Lévy process here, which is a process  $\{X(t), t \geq 0\}$  with increment  $X(t+s) - X(t)$  independent of the process  $\{X(v), 0 \leq v \leq t\}$  and has the same distribution law as  $X(s)$ , for every  $s, t \geq 0$ .

The generalized compound distribution may arise from practical situations as well. Consider the following manufacturing practice as an example in which processing time is a *Lévy Process with Exponential Horizon*. Suppose that the processing of a job is to erase the flaw of the job (or product). The flaw is an irregular area with length  $T$  and irregular width, which is a main reason that gives rise to the random processing time on erasing any fixed length of flaw area. The processing time to erase a flaw of  $l$  units of length is a random variable, related only to the length but unrelated to the position of the flaw. Then the processing time on a length of flaw is a Lévy process with the length as its time parameter. So if the length of flaw is distributed exponentially, then the processing time must be a Lévy process with exponential horizon.

We now calculate the characteristic function of  $X(T)$ . Denoting the characteristic exponent of  $X(1)$  by  $\Psi(t)$ , i.e.,  $\mathbf{E}[e^{isX(1)}] = e^{-\Psi(s)}$ , we have the following result.

**Proposition 2.2.** If  $\{X(t), t \geq 0\}$  is a Lévy process and  $T$  is exponentially distributed with rate  $\gamma$ , independent of  $\{X(t), t \geq 0\}$ , then the characteristic function of  $X(T)$  is

$$\phi_{X(T)}(s) = (1 + \Psi(s)/\gamma)^{-1}.$$

*Proof.* First note that a Lévy Process  $X(t)$  is infinitely divisible. Then the characteristic function of  $X(t)$  is given by  $\phi_{X(t)}(s) = e^{-t\Psi(s)}$  (see, for example, Bertoin, 1996). Now if we let  $T$  be exponentially distributed with rate  $\gamma$ , independent of the Lévy Process  $\{X(t), t \geq 0\}$ . Then the characteristic function of  $X(T)$  is

$$\begin{aligned} \phi_{X(T)}(s) &= \mathbf{E}[e^{isX(T)}] = \mathbf{E}\{\mathbf{E}[e^{isX(T)}|T]\} = E[e^{-T\Psi(s)}] \\ &= \gamma \int_0^\infty e^{-t\Psi(s)-\gamma t} dt = (1 + \Psi(s)/\gamma)^{-1}, \end{aligned}$$

which completes the proof. ■

Due to Examples 5 and 6, a class of distributions which share the common  $G(t)$  in (2.1) is referred to as a *class of compound type distributions*. We now give a lemma on this type of distributions, which will play a key role in the interchange procedures to find the optimal sequences for the scheduling problem. Suppose that  $P_i$ ,  $i = 1, 2, \dots, n$ , are mutually independent random variables with distribution functions  $F_i(x)$  respectively. Denote by  $F_{ij}(x)$  the distribution function of  $P_i + P_j$  and  $\phi_i(t)$  the characteristic function of  $P_i$ . If  $\phi_i(t)$  all have the form  $(a + G(t)/\gamma_i)^{-1}$  with a common  $G(t)$ , that would lead to some interesting properties. Exponential and geometric distribution are two obvious special cases of such a situation. For the following discussions, we will exclude the trivial case of  $G(t) = 0$ , which is equivalent to  $\phi_i(t) = 1$  and  $P_i \equiv 0$  for all  $i \in \{1, \dots, n\}$ .

**Lemma 2.1.** There exist a function  $G(t)$  and a series of numbers  $\gamma_1, \dots, \gamma_n$  such that  $\phi_i(t) = (1 + G(t)/\gamma_i)^{-1}$  if and only if

$$\frac{F_i(x) - F_{ij}(x)}{\gamma_i} = \frac{F_j(x) - F_{ij}(x)}{\gamma_j}.$$

Moreover,  $\gamma_i$  can take value  $1/\mathbf{E}[P_i]$ .

*Proof.* Since Fourier transformation is linear and the characteristic function of  $F_{ij}(x)$  is the product of the characteristic functions of  $F_i(x)$  and  $F_j(x)$ ,

$$\begin{aligned} \frac{F_i(x) - F_{ij}(x)}{\gamma_i} &= \frac{F_j(x) - F_{ij}(x)}{\gamma_j} \quad \text{for } 1 \leq i, j \leq n \\ \iff \frac{\phi_i(t) - \phi_i(t)\phi_j(t)}{\gamma_i} &= \frac{\phi_j(t) - \phi_i(t)\phi_j(t)}{\gamma_j} \quad \text{for } 1 \leq i, j \leq n \\ \iff \frac{\phi_i(t)}{\gamma_i(1 - \phi_i(t))} &= \frac{\phi_j(t)}{\gamma_j(1 - \phi_j(t))} \quad \text{for } 1 \leq i, j \leq n \quad (\text{denoted by } 1/G(t)) \\ \iff \phi_i(t) &= (1 + G(t)/\gamma_i)^{-1} \quad \text{for } 1 \leq i \leq n. \end{aligned} \quad (2.4)$$

Moreover, it is clear that the equivalence in (2.4) is still true if  $G(t)$  and  $\gamma_i$  are replaced with  $aG(t)$  and  $a\gamma_i$ , respectively, for any fixed  $a$ . Therefore, we can suppose  $G'(0) = -i$ . This together with  $i\mathbf{E}[P_i] = \phi'_i(0) = -(G'(0)/\gamma_i)/(1 + G(0)/\gamma_i)^2$  and  $G(0) = 0$  (as  $\phi_i(0) = 1$ ) leads to  $\gamma_i = 1/\mathbf{E}[P_i]$ . The proof is thus complete. ■

### 3. Optimal Policies without Precedences

Some orders between two nondecreasing functions are defined below to shorten notations. Suppose that  $H_1(x)$  and  $H_2(x)$  are two nondecreasing functions.

- $H_1(x)$  is said to be prior to  $H_2(x)$  in *increment order*, denoted as  $H_1 \prec_{inc} H_2$ , if  $H_2(x) - H_1(x)$  is nondecreasing in  $x$ , or equivalently,  $H_2(x)$  has greater increments than  $H_1(x)$  in the sense that

$$H_1(t) - H_1(s) \leq H_2(t) - H_2(s) \quad \text{for } t > s. \quad (3.1)$$

- $H_1$  is said to be prior to  $H_2$  in *convexity order*, denoted by  $H_1 \prec_{cv} H_2$  (or  $H_2 \succ_{cv} H_1$ ), if  $H_2(x)$  has more convexity than  $H_1(x)$  in the sense that

$$\begin{aligned} H_1(\alpha s + (1 - \alpha)t) - \alpha H_1(s) - (1 - \alpha)H_1(t) \\ \geq H_2(\alpha s + (1 - \alpha)t) - \alpha H_2(s) - (1 - \alpha)H_2(t) \end{aligned} \quad (3.2)$$

for all  $\alpha \in (0, 1)$ , which is equivalent to the convexity of  $H_2(x) - H_1(x)$ .

Let  $H_{ij}(t) = w_i\gamma_i m_i(t) - w_j\gamma_j m_j(t)$ . It follows from (3.1) and (3.2) that

$$w_i\gamma_i m_i \prec_{inc} w_j\gamma_j m_j \iff H_{ij}(t) \text{ is nonincreasing, and} \quad (3.3)$$

$$w_i\gamma_i m_i \prec_{cv} w_j\gamma_j m_j \iff H_{ij}(t) \text{ is concave.} \quad (3.4)$$

Now we come to the first theorem on model (1.1).

**Theorem 3.1.** Consider the problem  $1 \mid P_j \sim \phi_j(t) = (1 + G(t)/\gamma_j)^{-1} \mid \mathbf{E}[\sum f_j(C_j)]$ .

(1) When  $P_i$ ,  $i = 1, 2, \dots, n$ , are nonnegative, the sequence  $\{1, 2, \dots, n\}$  is optimal if

$$i > j \implies \gamma_i m_i \prec_{inc} \gamma_j m_j. \quad (3.5)$$

In other words, a sequence in non-increasing order  $\prec_{inc}$  of  $\{\gamma_i m_i(t)\}$  is optimal.

(2) When  $P_i$ ,  $i = 1, 2, \dots, n$ , are real-valued random variables with nonnegative means, the sequence  $\{1, 2, \dots, n\}$  is optimal if

$$i > j \implies \gamma_i m_i \prec_{inc} \gamma_j m_j \quad \text{and} \quad \gamma_i m_i \prec_{cv} \gamma_j m_j, \quad (3.6)$$

In other words, a sequence in non-increasing orders  $\prec_{inc}$  and  $\prec_{cv}$  of  $\{\gamma_i m_i(t)\}$  is optimal.

*Proof.* Let  $\pi = \{\dots, i, j, \dots\}$  be an arbitrary job sequence with  $i > j$ ,  $\pi' = \{\dots, j, i, \dots\}$  be the sequence by interchanging two consecutive jobs  $i, j$  in  $\pi$ , and  $C$  denote the completion time of the job prior to job  $i$  under  $\pi$ . Then, for the objective function  $ETC$  in (1.1), since  $\{f_i(t)\}$  are independent of  $\{P_i\}$ , we have

$$\begin{aligned} ETC(\pi) - ETC(\pi') &= \mathbf{E}[f_i(C + P_i)] + \mathbf{E}[f_j(C + P_i + P_j)] - \mathbf{E}[f_j(C + P_j)] - \mathbf{E}[f_i(C + P_i + P_j)] \\ &= \mathbf{E}[m_i(C + P_i)] + \mathbf{E}[m_j(C + P_i + P_j)] - \mathbf{E}[m_j(C + P_j)] - \mathbf{E}[m_i(C + P_i + P_j)]. \end{aligned}$$

Denote the cumulative distribution functions of  $P_i$ ,  $P_j$  and  $P_i + P_j$  by  $F_i(x)$ ,  $F_j(x)$  and  $F_{ij}(x)$  respectively as in Lemma 2.1. Since  $P_1, \dots, P_n$  are independent, conditional on  $C$ ,

$$\begin{aligned} ETC(\pi) - ETC(\pi') &= \mathbf{E}[m_i(C + P_i)] - \mathbf{E}[m_i(C + P_i + P_j)] - \mathbf{E}[m_j(C + P_j)] + \mathbf{E}[m_j(C + P_i + P_j)] \\ &= \mathbf{E}\left\{\int_{-\infty}^{\infty} m_i(C + x) d[F_i(x) - F_{ij}(x)]\right\} - \mathbf{E}\left\{\int_{-\infty}^{\infty} m_j(C + x) d[F_j(x) - F_{ij}(x)]\right\}. \end{aligned}$$

By Lemma 2.1,

$$\begin{aligned} ETC(\pi) - ETC(\pi') &= \frac{1}{\gamma_i} \mathbf{E}\left\{\int_{-\infty}^{\infty} (\gamma_i m_i(C + x) - \gamma_j m_j(C + x)) d[F_i(x) - F_{ij}(x)]\right\} \\ &= \frac{1}{\gamma_i} \left\{\mathbf{E}[H_{ij}(C + P_i)] - \mathbf{E}[H_{ij}(C + P_i + P_j)]\right\}. \end{aligned} \quad (3.7)$$

Now we consider two cases corresponding to parts (1) and (2) of the theorem.

**Case 1.**  $P_1, \dots, P_n$  are nonnegative variables. Under the condition that  $i > j$  implies  $\gamma_i m_i \prec_{inc} \gamma_j m_j$ ,  $H_{ij}(t)$  is non-increasing by (3.3). Hence it is clear by (3.7) that

$$ETC(\pi) - ETC(\pi') = \frac{1}{\gamma_i} \mathbf{E}\left\{H_{ij}(C + P_i) - H_{ij}(C + P_i + P_j)\right\} \geq 0. \quad (3.8)$$

**Case 2.**  $P_1, \dots, P_n$  are real-valued with  $E[P_i] \geq 0$ . Then under the condition that  $H_{ij}(x)$  are non-increasing and concave functions,

$$\begin{aligned} ETC(\pi) - ETC(\pi') &= \frac{1}{\gamma_i} \left\{ \mathbf{E}[H_{ij}(C + P_i) - H_{ij}(C + P_i + P_j)] \right\} \\ &\geq \frac{1}{\gamma_i} \left\{ \mathbf{E}[H_{ij}(C + P_i) - H_{ij}(C + P_i + \mathbf{E}P_j)] \right\} \geq 0, \end{aligned} \quad (3.9)$$

where the first inequality follows from applying Jensen's inequality to concave function  $H_{ij}(x)$  conditional on  $C + P_i$ , and the second inequality holds as  $H_{ij}(x)$  is non-increasing.

(3.8) and (3.9) show that the expected cost is reduced by interchanging the position of job  $i$  and  $j$  in sequence  $\pi = \{\dots, i, j, \dots\}$  with  $i > j$ . Thus the theorem is proved ■

We now discuss cost function (1.2). Application of Theorem 3.1 yields following theorem for the problem  $1 \mid P_j \sim \phi_j(t) = (1 + G(t)/\gamma_j)^{-1}, D_j \sim F_j \mid \mathbf{E}[\sum w_j g(C_j - D_j)]$ .

**Theorem 3.2.** Suppose that  $\{D_i\}$  have a common distribution. Then a sequence in non-increasing order of  $\{\gamma_i w_i\}$ , or equivalently, in non-decreasing order of  $\{\mathbf{E}(P_i)/w_i\}$ , minimizes the  $ETC(\pi)$  in (1.2) if either

- (1)  $g(t)$  is an increasing function and  $P_i$ 's are nonnegative, or,
- (2)  $g(t)$  is a convex and non-decreasing function and  $P_i$ 's are real-valued with nonnegative means.

*Proof.* We first note by Lemma 2.1 that non-increasing order of  $\{\gamma_i w_i\}$  is equivalent to non-decreasing order of  $\{\mathbf{E}(P_i)/w_i\}$  since  $\gamma_j$  may take  $1/\mathbf{E}[P_j]$ . Let  $f_i(t) = w_i g(t - D_i)$ ,  $i = 1, 2, \dots, n$ , and  $D$  be a representative of  $D_i$ 's. Then

$$m_i(t) = \mathbf{E}[f_i(t)] = w_i \mathbf{E}[g(t - D_i)] = w_i \mathbf{E}[g(t - D)],$$

which gives  $H_{ij} = (w_i \gamma_i - w_j \gamma_j) \mathbf{E}[g(t - D)]$ . Since  $g(t)$  is non-decreasing,

$$\gamma_i m_i \prec_{inc} \gamma_j m_j \iff H_{ij}(t) \text{ is non-increasing} \iff w_i \gamma_i \leq w_j \gamma_j. \quad (3.10)$$

Thus if  $\gamma_1 w_1 \geq \dots \geq \gamma_n w_n$ , then  $\{1, \dots, n\}$  is optimal by Theorem 1 and a sequence in non-increasing order of  $\{\gamma_i w_i\}$  minimizes  $ETC(\pi)$ . This proves part (1).

To prove part (2), note that  $g$  is convex, which implies that

$$\gamma_i m_i \prec_{cv} \gamma_j m_j \iff H_{ij}(x) \text{ is concave} \iff w_i \gamma_i \leq w_j \gamma_j. \quad (3.11)$$

Combine (3.11) with (3.10), part (2) is proved. ■

**Remark 3.1.** Theorem 3.2 reveals an interesting fact when we consider the problem of minimizing the expected discounted cost function, i.e., problem  $1 \mid P_j \sim \phi_j(t) = (1 + G(t)/\gamma_j)^{-1} \mid \mathbf{E}[\sum w_j (1 - e^{-rC_j})]$ . This problem is a special case of the model in Theorem 3.2 by setting  $D_i$  to 0 and taking  $g(t) = 1 - e^{-rt}$ . It is well-known that the sequence with

weighted discounted shortest expected processing time first rule (WDSEPT) is optimal for problem 1 ||  $\mathbf{E}[\sum w_j(1 - e^{-rC_j})]$ . Theorem 3.2 says that when the distributions of processing times belong to the class of distributions with characteristic functions  $(1 + G(t)/\gamma)^{-1}$ , the WDSEPT rule reduces to the WSEPT rule (weighted shortest expected processing time first).

The following Theorems 3.3 and 3.4 deal with model (1.2) with different due dates, for nonnegative and real-valued processing times respectively.

**Theorem 3.3.** Let  $g$  be a non-decreasing function. The sequence  $\{1, 2, \dots, n\}$  minimizes the  $ETC(\pi)$  in (1.2) if

$$\gamma_1 w_1 F_1 \succ_{inc} \gamma_2 w_2 F_2 \succ_{inc} \dots \succ_{inc} \gamma_n w_n F_n. \quad (3.12)$$

Furthermore, if  $g$  is also concave (convex), then  $\{1, 2, \dots, n\}$  is optimal if

$$\{\gamma_i w_i S_i(x), i = 1, \dots, n\} \text{ is non-increasing (non-decreasing)}. \quad (3.13)$$

*Proof.* Let  $f_i(t) = w_i g(t - D_i)$  then

$$m_i(t) = w_i \mathbf{E}[g(t - D_i)] = w_i \int_0^\infty g(t - x) dF_i(x),$$

which yields

$$H_{ij}(t) = \int_0^\infty g(t - x) d[\gamma_i w_i F_i(x) - \gamma_j w_j F_j(x)]. \quad (3.14)$$

For  $i > j$ , since  $\gamma_i w_i F_i \prec_{inc} \gamma_j w_j F_j$ ,  $\gamma_i w_i F_i(x) - \gamma_j w_j F_j(x)$  is non-increasing in  $x$ . Hence,  $H_{ij}(t)$  is non-increasing by (3.14) for  $g(t - x)$  is non-decreasing in  $t$ , which is equivalent to  $\gamma_i m_i \prec_{inc} \gamma_j m_j$ . The first part of the theorem then follows from Theorem 3.1.

We now turn to the second part. The cases of  $g$  being concave and convex are considered separately.

**Case 1** (concave  $g$ ). Concavity of  $g$  implies that it is continuous. By (3.14) and iterated integration,

$$H_{ij}(t) = g(t)(\gamma_i w_i - \gamma_j w_j) + \int_0^\infty [\gamma_i w_i S_i(x) - \gamma_j w_j S_j(x)] dg(t - x).$$

Hence for  $t > s$ ,

$$\begin{aligned} H_{ij}(t) - H_{ij}(s) &= (g(t) - g(s))(\gamma_i w_i - \gamma_j w_j) \\ &\quad + \int_0^\infty [\gamma_i w_i S_i(x) - \gamma_j w_j S_j(x)] d[g(t - x) - g(s - x)]. \end{aligned} \quad (3.15)$$

Since  $g$  is concave, for arbitrary  $\epsilon > 0$ ,

$$\begin{aligned} &[g(t - (x + \epsilon)) - g(s - (x + \epsilon))] - [g(t - x) - g(s - x)] \\ &= [g(s - x) - g(s - x - \epsilon)] - [g(t - x) - g(t - x - \epsilon)] \geq 0. \end{aligned} \quad (3.16)$$

That is,  $g(t-x) - g(s-x)$  is non-decreasing in  $x$ . Obviously,  $\gamma_i w_i S_i(x) \leq \gamma_j w_j S_j(x)$  implies  $\gamma_i w_i \leq \gamma_j w_j$  (by taking  $x = 0$ ). By (3.15), if  $g$  is concave and non-decreasing, then  $H_{ij}(t) - H_{ij}(s) \leq 0$ , i.e.,  $w_i m_i \prec_{inc} w_j m_j$ . The theorem is thus proved for concave  $g$ .

**Case 2** (convex  $g$ ). The fact that  $g$  is a convex gives that  $g(x)$  has finite right derivative, denoted as  $g'_+(x)$ , at every point  $x$ . Moreover, we have again by the convexity of  $g$  that

$$g(t-x) - g(s-x) \geq (t-s)g'_+(s-x) \geq 0, \quad (3.17)$$

which implies that  $g(t-x) - g(s-x)$  is bounded from below with respect to  $x$  for arbitrary  $t > s$ . Hence (3.15) can be rewritten as

$$H_{ij}(t) - H_{ij}(s) = - \int_0^\infty [\gamma_i w_i F_i(x) - \gamma_j w_j F_j(x)] d[g(s-x) - g(t-x)].$$

As  $g$  is convex, analogous to (3.16), we see that  $g(t-x) - g(s-x)$  is non-increasing in  $x$ . So if  $\gamma_i w_i F_i \leq \gamma_j w_j F_j$ , then  $H_{ij}(t) - H_{ij}(s) \leq 0$  for  $t > s$ , i.e.,  $w_i m_i \prec_{inc} w_j m_j$ , which completes the proof of the theorem. ■

The following theorem is for the case of real-valued  $P_i$  with nonnegative means. We omit its proof, which is similar to that of Theorem 3.3.

**Theorem 3.4.** Let  $P_1, \dots, P_n$  be real-valued random variables with nonnegative means and  $g$  non-decreasing. Then the sequence  $\{1, 2, \dots, n\}$  minimizes the  $ETC(\pi)$  in (1.2) if either

- (1)  $g$  is convex and  $\gamma_1 w_1 F_1 \succ_{inc} \gamma_2 w_2 F_2 \succ_{inc} \dots \succ_{inc} \gamma_n w_n F_n$ , or,
- (2)  $g$  is derivable with  $g'(x)$  being convex and  $\gamma_1 w_1 F_1(x) \geq \gamma_2 w_2 F_2(x) \geq \dots \geq \gamma_n w_n F_n$ .

## 4. Scheduling with Precedence Constraints of Nonpreemptive Chains

In this section we consider scheduling with precedence constraints. Suppose that jobs  $\{1, \dots, n\}$  are divided into  $m$  nonpreemptive chains  $u_i = \{i_1, \dots, i_{k_i}\}$ ,  $i = 1, \dots, m$ . The processing of the jobs within any chain follows the increasing order of its indices and the switch to another chain is disallowed. That is, once the first job of a chain, say  $u_{i1}$ , is selected to process on the machine, the processing on a job in another chain is disallowed until all jobs in the same chain are finished. Denote the scheduling problems with cost functions (1.1) and (1.2) under precedence constraints respectively by  $1 \mid P_j \sim \phi(t) = (1 + G(t)/\gamma_j)^{-1}$ , chains  $\mid \mathbf{E}[\sum f_j(C_j)]$  and  $1 \mid P_j \sim \phi(t) = (1 + G(t)/\gamma_j)^{-1}$ ,  $D_j \sim F_j$ , chains  $\mid \mathbf{E}[\sum w_j g_j(C_j - D_j)]$ .

Corresponding to the  $i$ th chain  $u_i$ , we denote  $M_i = \{m_{ij}(t), j \in u_i\}$ ,  $i = 1, \dots, m$ . Let  $H = \{h_1, \dots, h_m\}$  and  $G = \{g_1, \dots, g_n\}$  be two groups of functions. We say that  $H \prec_{inc} G$  if  $h_i \prec_{inc} g_j$  for  $1 \leq i \leq m, 1 \leq j \leq n$ . Similarly,  $H \prec_{cv} G$  if  $h_i \prec_{cv} g_j$  for  $1 \leq i \leq m, 1 \leq j \leq n$ . The following theorem provides an optimal sequence for the scheduling problem with precedence constraints.



**Theorem 4.1.** Consider problem 1 |  $P_j \sim \phi(t) = (1 + G(t)/\gamma_j)^{-1}$ , chains |  $\mathbf{E}[\sum f_j(C_j)]$ .

- (1) Suppose that  $P_i$ ,  $i = 1, 2, \dots, n$ , are nonnegative. The sequence  $\{u_1, \dots, u_m\}$  is optimal if  $i > j$  implies that  $M_i$  is prior to  $M_j$  in order  $\prec_{inc}$ , i.e.,  $M_i \prec_{inc} M_j$ .
- (2) Suppose that  $P_i$ ,  $i = 1, 2, \dots, n$ , are real-valued with nonnegative means. The sequence  $\{u_1, \dots, u_m\}$  is optimal if  $i > j$  implies that  $M_i$  is prior to  $M_j$  in orders  $\prec_{inc}$  and  $\prec_{cv}$ , i.e.,  $M_i \prec_{inc} M_j$  and  $M_i \prec_{cv} M_j$ .

*Proof.* Let  $\pi = \{\dots, u, v, \dots\}$ , a permutation of job chains  $\{u_1, \dots, u_m\}$ , and  $\pi' = \{\dots, v, u, \dots\}$ , a job sequence of chains by interchanging two consecutive chains  $u$  and  $v$ , where  $u = \{u_1, \dots, u_s\}$  and  $v = \{v_1, \dots, v_r\}$ . Denote by  $C$  the completion time of the job immediately preceding the first job of job chain  $u_1$  under sequence  $\pi$ . We also write for convenience that

$$P_u^i = \sum_{k=0}^i P_{u_k}, \quad P_v^j = \sum_{k=0}^j P_{v_k} \text{ and } P_{u_0} = P_{v_0} \equiv 0.$$

Then

$$\begin{aligned} ETC(\pi) - ETC(\pi') &= \mathbf{E} \left[ \sum_{i=1}^r m_{u_i}(C + P_u^i) + \sum_{j=1}^s m_{v_j}(C + P_u^r + P_v^j) \right] \\ &\quad - \mathbf{E} \left[ \sum_{j=1}^s m_{v_j}(C + P_v^j) + \sum_{i=1}^r m_{u_i}(C + P_v^s + P_u^i) \right] \\ &= \mathbf{E} \sum_{i=1}^r [m_{u_i}(C + P_u^i) - m_{u_i}(C + P_u^i + P_v^s)] \\ &\quad - \mathbf{E} \sum_{j=1}^s [m_{v_j}(C + P_v^j) - m_{v_j}(C + P_u^r + P_v^j)] \\ &= \sum_{i=1}^r \sum_{j=1}^s \mathbf{E} \left\{ [m_{u_i}(C + P_u^i + P_v^{j-1}) - m_{u_i}(C + P_u^i + P_v^j)] \right. \\ &\quad \left. - [m_{v_j}(C + P_u^{i-1} + P_v^j) - m_{v_j}(C + P_u^i + P_v^j)] \right\}. \end{aligned} \quad (4.1)$$

Conditional on  $C + P_u^{i-1} + P_v^{j-1}$ , Lemma 2.1 again yields

$$\begin{aligned} &\mathbf{E} [m_{v_j}(C + P_u^{i-1} + P_v^j) - m_{v_j}(C + P_u^i + P_v^j)] \\ &= \frac{\gamma_{v_j}}{\gamma_{v_j}} \mathbf{E} [m_{u_i}(C + P_u^i + P_v^{j-1}) - m_{v_j}(C + P_u^i + P_v^j)]. \end{aligned}$$

(4.1) can thus be rewritten as

$$ETC(\pi) - ETC(\pi') = \sum_{i=1}^r \sum_{j=1}^s \frac{1}{\gamma_{ui}} \mathbf{E} [H_{u_i v_j}(C + P_u^i + P_v^{j-1}) - H_{u_i v_j}(C + P_u^i + P_v^j)], \quad (4.2)$$

which implies the theorem. ■

Almost the same arguments give the following theorems. Let  $A$  and  $B$  be two sets of real numbers or functions. We say that  $A \geq_G B$  if  $x \geq y$  holds for all  $x \in A$  and  $y \in B$ .

**Theorem 4.2.** Denote  $\Gamma_i = \{w_j \gamma_j | j \in u_i\}$  and suppose  $D_i$  have a common distribution, then the sequence  $\{u_1, \dots, u_m\}$ , in non-increasing order of  $\Gamma_i$  with respect to order  $\leq_G$ , minimizes the  $ETC(\pi)$  in (1.2) if

- (1)  $g(t)$  is a non-decreasing function and the processing times  $P_i$ 's are nonnegative, or,
- (2)  $g(t)$  is a convex and non-decreasing function and the processing times  $P_i$ 's are real valued with nonnegative mean.

**Theorem 4.3.** Let  $\mathcal{F}_i = \{w_j \gamma_j F_j | j \in u_i\}$  and  $\mathcal{S}_i = \{w_j \gamma_j S_j | j \in u_i\}$ . Then the sequence  $\{u_1, u_2, \dots, u_m\}$  minimizes the  $ETC(\pi)$  in (1.2) if  $\mathcal{F}_1 \succ_{inc} \mathcal{F}_2 \succ_{inc} \dots \succ_{inc} \mathcal{F}_n$ , provided that  $g$  in objective function (1.2) is non-decreasing. Moreover, if  $g$  is also concave (or convex), then  $\{u_1, u_2, \dots, u_m\}$  is optimal provided that  $\mathcal{S}_1(\mathcal{F}_1) \geq_G \dots \geq_G \mathcal{S}_m(\mathcal{F}_m)$ .

**Theorem 4.4.** Let  $P_1, \dots, P_n$  be real-valued random variables with nonnegative means. Then the sequence  $\{1, 2, \dots, n\}$  minimizes the  $ETC(\pi)$  in (1.2) if  $\mathcal{F}_1 \succ_{inc} \mathcal{F}_2 \succ_{inc} \dots \succ_{inc} \mathcal{F}_n$  when  $g$  is convex and non-decreasing, or if  $\mathcal{F}_1(x) \geq_G \mathcal{F}_2(x) \geq_G \dots \geq_G \mathcal{F}_n(x)$  when  $g$  is differentiable with convex  $g'(x)$ .

## 5. Concluding Remarks

In this paper we generalized previous studies on stochastic single-machine scheduling to more general situations. In particular, the model considered by Cai and Zhou (2005) allows general stochastic cost functions that subsume many performance measures commonly studied in the literature, but was restricted to exponential processing times only. We have extended the distributions of processing times to a much larger class than the exponential distributions, which includes, among others, the compound geometric distributions and processes with random horizon. These more general distributions can arise naturally from practical scheduling problems.

We have derived optimal sequences to minimize the expected total cost with general stochastic cost functions in a number of situations based on increment order as well as convexity order, which generalize previous results in the literature. In addition, we have considered the scheduling problem with precedence constraints as well.

Interesting future research work may include different and random job arrival times, stochastic weights as functions of time, the preemptive version of the problem, stochastic machine breakdowns, multiple machine scheduling, and dynamic policies.

## References

- [1] Bagga, P. C. and K. R. Kalra. (1981). Single machine scheduling problem with quadratic functions of completion times - a modified approach. *Journal of Inform. Optim. Sci.*, **2**, 103-108.
- [2] Bertoin, J. (1996). *Lévy Process*, Cambridge University Press.

- [3] Bisgaard T.M. and S. Zoltn. (2000). Characteristic functions and moments sequences: positive definiteness in probability, Nova science publishers, Inc.
- [4] Boxma, O.J., and F.G. Forst. (1986). Minimizing the expected weighted number of tardy jobs in stochastic flow shops. *Operations Research Letters*. **5**, 119-126.
- [5] Boys, R.J., K.D. Glazebrook, C.M. McCrone (1997) Single machine scheduling when processing times are correlated normal random variables. *European Journal of Operation Research* **102**, 111-123
- [6] X. Cai and X. Zhou. (2000). Asymmetric earliness and tardiness scheduling with exponential processing times on an unreliable machine. *Annals of Operations Research*. **98**, 313-331. 2000.
- [7] X. Cai and X. Zhou. (2005). Single-machine scheduling with exponential processing time and general stochastic cost functions. *Journal of Global Optimization*, to appear.
- [8] Derman, C., G. Lieberman and S. Ross. (1978). A renewal decision problem. *Management Science*, **24**, 554-561.
- [9] Glazebrook, K.D. (1979). Scheduling tasks with exponential service times on parallel processors. *Journal of Applied Probability*. **16**, 658-689.
- [10] Jang, Wooseung, Cerry M. Klein. (2002) Minimizing the expected number of tardy jobs when processing times are normally distributed *Operation research letters*. **30**, 100-106
- [11] Kampke, T. (1989). Optimal scheduling of jobs with exponential service times on identical parallel processors. *Operations Research*. **37**, 126-133.
- [12] Pinedo, M. (1983). Stochastic scheduling with release dates and due dates. *Operations Research*. **31**, 559-572.
- [13] Pinedo, M. (1995). *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs.
- [14] Rothkopf, M.H., (1966). Scheduling with random service times, *Management Science*, **12**, 437-447.
- [15] Righter, R. (1994). Scheduling. In M. Shacked and J. G. Shanthikumar (Eds.): *Stochastic Orders and Their Applications*, Academic Press, Boston. 381-428.
- [16] Sarin, S.C., E. Erel, and G. Steiner. (1991). Sequencing jobs on a single machine with a common due date and stochastic processing times. *European Journal of Operational Research*. **51**, 287-302.
- [17] Weiss, G., and M. Pinedo, (1980). Scheduling tasks with exponential services times on non-identical processors to minimize various cost functions. *Journal of Applied Probability*. **17**, 187-202.

# A COMPARISON OF DIFFERENT FORMULATIONS FOR THE NON-PREEMPTIVE SINGLE MACHINE TOTAL WEIGHTED TARDINESS SCHEDULING PROBLEM

Ketan Khowala, Ahmet Keha and John Fowler  
*Arizona State University, Tempe, USA*

**Abstract:** In this paper, the computational performance of four different formulations for the single machine total weighted tardiness (SMTWT) problems is studied. These formulations are based on start and completion time variables, time index variables, linear ordering variables and positional assignment variables. Experimentation documented in this paper leads to the conclusion that the most appropriate formulation depends on the sum of the processing times of the jobs.

**Key words:** Integer programming, Total weighted tardiness, Scheduling.

## 1. INTRODUCTION

In this paper, we examine the computational performance of four different formulations for the single machine total weighted tardiness (SMTWT) problem. The formulations are classified based on the choice of the decision variables. The choice of decision variables determines how the objective function and the constraints can be modeled. This paper compares four different formulations of SMTWT with (i) start time and completion time variables [Balas (1985)] (ii) time index variables [Sousa and Wolsey (1992)] (iii) linear ordering variables [Dyer and Wolsey (1990)] and (iv) assignment and positional date variables [Lasserre and Queyranne (1992)]. Queyranne and Schulz (1994) provide a survey of polyhedral approaches to machine scheduling problems and also compiled the various formulations for single machine scheduling problems. Through lot of work has been done in defining different kinds of formulations in literature, a computational study to compare these formulations is needed. The objective of our problem is  $\sum w_j T_j$ , where  $w_j$  denotes the weights associated with the jobs and  $T_j$  denotes the tardiness of jobs. Let  $p_j$  be the processing time of job  $j$ ,  $d_j$  the due date of the job  $j$  and  $C_j$  the completion time of job  $j$ . The tardiness is defined as  $T_j = \max(C_j - d_j, 0)$ . If  $n$  is the total number of jobs, then the set of jobs  $N$  is defined as  $N \in \{1, 2, \dots, n\}$ . In the scheduling notation of Graham *et al.* (1979), the SMTWT problem is defined as  $1 || \sum w_j T_j$ .

## 2. MIP FORMULATIONS AND LITERATURE REVIEW

In this section we present different mixed integer programming (MIP) formulations for the  $1||\sum w_j T_j$  problem.

### 2.1 Start Time and Completion Time Variables [F1]

In this formulation, we let binary variable  $y_{jk}$  equal to one if job  $j$  is processed before job  $k$  and equal to zero otherwise. Completion time constraints and the disjunctive constraints are needed for formulating single machine scheduling problems using completion time variables. These same constraints could also be written in terms of start time variables.

$$\text{Objective: Min } \sum w_j T_j \quad [F1]$$

$$C_j \geq p_j \quad \forall j \in N \quad (1.1)$$

$$C_j + p_k \leq C_k + M(1 - y_{jk}) \quad \text{for } j < k \text{ \& } j, k \in N \quad (1.2)$$

$$C_k + p_j \leq C_j + M y_{jk} \quad \text{for } j < k \text{ \& } j, k \in N \quad (1.3)$$

$$T_j \geq C_j - d_j \quad \forall j \in N \quad (1.4)$$

$$T_j \geq 0 \quad \forall j \in N \quad (1.5)$$

$$C_j \geq 0 \quad \forall j \in N \quad (1.6)$$

$$y_{jk} \in \{0,1\} \quad \forall j, k \in N \quad (1.7)$$

Constraint set (1.1) ensures that the completion time of each job is greater than or equal to the processing time of the job. Constraint sets (1.2) and (1.3) are disjunctive constraints which enforce that either job  $j$  is before job  $k$  or job  $k$  is before job  $j$  for each pair of jobs. Constraint sets (1.4) and (1.5) are the tardiness constraints, defining  $T_j$  as  $\max(C_j - d_j, 0)$ . Further, constraint sets (1.6) and (1.7) are the non-negativity and integrality constraints. In this formulation, the value of big  $M$  is taken to be greater than the sum of the processing times of all the jobs.

### 2.2 Time Index Variables [F2]

In the time index variable formulation, the planning horizon,  $T$  is discretized into the periods 1, 2, 3, ...,  $T$ , where period  $t$  starts at time  $t-1$  and ends at time  $t$ . A binary time index variable is introduced, where  $x_{jt}$  is one if job  $j$  starts at time  $t$ , is equal to zero otherwise. The formulation with time index variable is as follows:

$$\text{Objective: Min } \sum w_j T_j \quad [F2]$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \quad \forall j \in N \quad (2.1)$$

$$\sum_{j=1}^n \sum_{s=\max(0, t-p_j+1)}^t x_{js} \leq 1 \quad t = 1, \dots, T \quad (2.2)$$

$$T_j \geq \sum_{t=1}^{T-p_j+1} (t-1 + p_j) x_{jt} - d_j \quad \forall j \in N \quad (2.3)$$

$$T_j \geq 0 \quad \forall j \in N \quad (2.4)$$

$$x_{jt} \in \{0,1\} \quad \forall j \in N \text{ \& } t = 1, \dots, T \quad (2.5)$$

The first constraint set (2.1) enforces that each job can be started only at one particular time and the second constraint set (2.2) ensures that at any given time, a maximum of only one job can be started. Using the time index variable formulation, the completion time of a job  $j$  can be expressed as  $\sum_{t=1}^{T-p_j+1} (t-1+p_j)x_{jt}$ . The constraint sets (2.3), (2.4) and (2.5) serve the same purpose as constraint sets (1.4) – (1.7) in F1.

### 2.3 Linear Ordering Variable [F3]

We define the linear ordering variables  $d_{jk}$  for  $j, k \in N$  as binary variables which is equal to one when job  $j$  precedes job  $k$  and is equal to zero otherwise. The formulation with linear ordering variables is as follows:

$$\text{Objective: Min } \sum w_j T_j \quad [\text{F3}]$$

$$d_{jk} + d_{kj} = 1 \quad 1 \leq j \leq k \leq n \quad (3.1)$$

$$d_{jk} + d_{kl} + d_{lj} \leq 2 \quad j \neq k \neq l \neq j, \quad j, k, l \in N \quad (3.2)$$

$$d_{jk} \geq 0 \quad j \neq k \text{ \& } j, k \in N \quad (3.3)$$

$$T_j \geq \sum_{\substack{k \in N \\ k \neq j}} p_k d_{kj} + p_j - d_j \quad \forall j \in N \quad (3.4)$$

$$T_j \geq 0 \quad \forall j \in N \quad (3.5)$$

$$d_{jk} \in \{0,1\} \quad \forall j, k \in N \quad (3.6)$$

Constraint set (3.1) is a set of conflict constraints, which ensure that either job  $j$  is processed before job  $k$  or job  $k$  is processed before job  $j$ . Constraint set (3.2) represents the transitivity constraints that ensure a linear order between three jobs. The completion time of job  $j$  can be expressed as  $\sum_{\substack{k \in N \\ k \neq j}} p_k d_{kj} + p_j$

when using linear ordering variables. Hence the tardiness constraints are enforced using sets (3.4) and (3.5).

### 2.4 Assignment and Positional Date Variables [F4]

In this formulation, we make use of assignment binary variables  $u_{jk}$ , which is equal to one if job  $j$  is assigned to position  $k$  and is equal to zero otherwise. Further, to complete the formulation, we also need positional date variables, which define either the start time or the completion time of the job at position  $k$ . In our formulation we use  $g_k$  to denote the completion time of the  $k^{\text{th}}$  job in the schedule or in other words, the completion time of the job at position  $k$ .

$$\text{Objective: Min } \sum w_j T_j \quad [\text{F4}]$$

$$\sum_{k \in N} u_{jk} = 1 \quad \forall j \in N \quad (4.1)$$

$$\sum_{j \in N} u_{jk} = 1 \quad \forall k \in N \quad (4.2)$$

$$g_k \geq g_{k-1} + \sum_j p_j u_{jk} \quad k = 2, \dots, N \quad (4.3)$$

$$g_1 = \sum_j p_j u_{j1} \quad (4.4)$$

$$C_j \geq g_k - M(1 - u_{jk}) \quad \forall j, k \in N \quad (4.5)$$

$$T_j \geq C_j - d_j \quad \forall j \in N \quad (4.6)$$

$$T_j \geq 0 \quad \forall j \in N \quad (4.7)$$

$$C_j \geq 0 \quad \forall j \in N \quad (4.8)$$

$$g_k \geq 0 \quad \forall k \in N \quad (4.9)$$

$$u_{jk} \in \{0,1\} \quad \forall j, k \in N \quad (4.10)$$

The constraint sets (4.1) and (4.2) ensure that a particular job is positioned only once and each position is only assigned to one job. Constraint sets (4.3) and (4.4) are the completion time constraints for position  $k$ . These can also be expressed in terms of the start time variables of the job at position  $k$ . The next constraint set (4.5), is for the completion time of job  $j$ . Constraints (4.6) - (4.8) are similar to (1.4) – (1.7) in F1.

In this formulation, the value of big  $M$  is taken to be greater than the sum of the processing times of all the jobs.

### 3. COMPUTATIONAL RESULTS

#### 3.1 Data Sets

We run our experiments for different set of parameters for all of these formulations. We adopt the idea presented by Hariri and Potts (1983), Potts and Van Wassenhove (1982, 1983), and Abdul-Razaq, Potts and Van Wassenhove (1990). Further this method of parameter generation is cited and adopted by many other authors in their computational experimentation.

An integer processing time  $p_j$  is generated from two different uniform distributions  $[1, 100]$  and  $[1, 10]$  (for two different test cases) and an integer weight  $w_j$  is generated from the uniform distribution  $[1, 10]$ . Based on a personal communication with Queyranne (2004), we decided to study both sets of processing time distributions, since the performance of F2 is known to be highly influenced by the sum of the processing times.

The due date  $d_j$  is given by the integer sample generated from the uniform distribution  $[P(L-R/2), P(L+R/2)]$ , where  $P$  is the sum of processing times of all the jobs and the two parameters  $L$  and  $R$ , are a relative measure of the location and range of the distribution, respectively. We chose  $L$  to be 0.5 and  $R$  is taken from the set  $R \in \{0.4, 0.8, 1.4\}$ . We run 3 problem instances for each of the 3 different values of  $R$ , generating total of 9 runs for each of the four different formulations for this problem. The number of jobs  $n$  is chosen from the set  $n \in \{20, 40, 60, 100\}$ . Some initial computations were conducted and we found that the results obtained with  $L=0.7$  and  $L=0.5$  are very much similar except that there were cases with

zero objective values for  $L=0.7$ . The test cases generated with  $L=0.5$  are much harder to solve and hence this value was selected for comparison of computational efficiency for this paper.

### 3.2 Results

As proposed in section 3.1, we conduct our experimentation for different formulations. The results for each different number of jobs, different formulations and for two different distribution of processing times are based on 9 different problem instances solved with different parameter settings of  $L$  and  $R$ . These formulations are modeled using AMPL and CPLEX version 8.1 is used to solve the generated problem instances. The experiments were run on a Linux distributed machine with 2.4 GHz processor and 1GB memory.

To compare the different formulations arising from the choice of different variables, we look at the objective function value of an LP relaxation, the computational time for the problem instances for which the optimal solution is obtained, the number of nodes explored, the number of tests cases unsolved in one hour, and the optimality gap for the instances with a feasible solution. The formulations are also compared based on the number of test instances (out of 9 experiments for a particular number of jobs) either the optimal or the best feasible solution is obtained for each formulation.

The results are presented in Tables 1 and 2. Results with processing time  $U [1, 100]$  are presented in Table 1, while the results with processing time  $U [1, 10]$  are presented in Table 2.

The findings from the computations, observed in Table 1 include:

- All formulations have difficulty as the number of jobs increases.
- Formulations 1 and 2 (F1 and F2), do not produce optimal solution as frequently as formulations 3 and 4 (F3 and F4). It is obvious F2 is not good because as the number of jobs increases, we are not even able to solve the LP relaxation. F1 solves the LP relaxation faster, but that does not help us very much as the bound is not tight.
- F4 often finds a feasible solution, but the optimality gap is higher than for F3, as the lower bound obtained by F4 is not as tight as the lower bound obtained from F3.
- F4 is better in solving the LP relaxation faster as the number of nodes explored is more, hence F4 might be good for developing other solution techniques (e.g. branch and cut and LP based heuristics).

The findings from the computations, observed in Table 2 are summarized as follows:

- F1 is clearly inferior to all the other formulations.
- F3 is relatively the same as it was in Table 1, as the constraint set does not change by changing to the lower range of processing times. Only the value of the objective function is affected.
- F1 and F4 perform better with the lower range of processing times. Since the value of big  $M$  in F1 and F4 depends on the sum of processing times of all the jobs, which in turn depends on the mean processing times of the jobs, narrowing this range helps in reducing the value of big  $M$ .
- F2 performs the best among all the formulations with the lower range of processing times. The LP relaxation is much easier to solve and also the bounds obtained are tighter. This is because the size of the constraint set of F2 is a function of the mean processing times of the jobs.



## 4. CONCLUSION AND FUTURE WORK

Based on the computational results, it can be concluded that the performance of these formulations very much depends on the sum of the processing times of all the jobs. F2 and F3 appear to be the most widely used formulations in Integer Programming and Scheduling literature and F4 appears to be the least widely used. While F1 often appears in textbooks and other literature that simply describe the problem (not the solution methodology) and it clearly does not generally perform well in practice.

F2 and F3 have been preferred due to the fact that they generally produce tighter bounds. However, we have found that the LP relaxation of these formulations tend to be much more difficult to solve. This is particularly true for F2 when  $P$  (sum of processing time of all the jobs) is large. The time required to solve the LP relaxation for these formulations means that fewer nodes of the branch and bound tree can be explored for a fixed computational budget. This limits one's ability to explore recent advancement in IP methodology (such as branch and cut). On the other hand, the LP relaxation of F4 can be solved relatively quickly, so these techniques offer some promise for these new advanced techniques. We note that when  $P$  is small, F2 becomes the preferred formulation. Thus these computational results provide us significant insights about the behavior of the four different formulations and an opportunity to explore them further to improve their efficiency.

Further, some of the other non preemptive single machine scheduling problems (like total weighted competition time, maximum lateness, etc.) could also be formulated and their computational efficiencies could be compared. This could also be extended to problems with release dates. A comprehensive study of the computational aspects of all of these formulations for different types of single and parallel machine scheduling problems could provide additional insights into these formulations.

## REFERENCE

- T. S. Abdul-Razaq, C. N. Potts and L. N. Van Wassenhove, 'A survey of algorithms for the single machine total weighted tardiness scheduling problem', *Discrete Applied Mathematics*, 26, 1990, 235-253.
- E. Balas, 'On the facial structure of scheduling polyhedra', *Mathematical Programming*, 24, 1985, 179-218.
- M. E. Dyer and L. A. Wolsey, 'Formulating the single machine sequencing problem with release dates as a mixed integer program', *Discrete Applied Mathematics*, 26 (1990), 255-270.
- 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*, 5 (1979), 287-326.
- A. M. A. Hariri and C. N. Potts, 'An algorithm for single machine sequencing with release dates to minimize total weighted completion time', *Discrete Applied Mathematics*, 5, 1983, 99-109.
- J. B. Lasserre and M. Queyranne, 'Generic scheduling polyhedral and a new mixed-integer formulation for single-machine scheduling', *Proceedings of the Second IPCO Conference, Carnegie-Mellon University, Pittsburgh*, 1992, 136-149.
- C. N. Potts and L. N. Van Wassenhove, 'A decomposition algorithm for the single machine, total tardiness problem', *Operations Research Letters*, 1, 1982, 177-181.
- C. N. Potts and L. N. Van Wassenhove, 'An algorithm for single machine sequencing with deadlines to minimize total weighted completion time', *European Journal of Operations Research*, 12, 1983, 379-389.
- M. Queyranne, Personal Communication at INFORMS Annual Conference, Denver, October, 2004.

- M. Queyranne and A. S. Schulz, 'Polyhedral approaches to machine scheduling', Technical Report 408/1994, Department of Mathematics, Technical University of Berlin, Berlin, Germany, 1994.
- J. P. Sousa and L. A. Wolsey, 'A time-indexed formulation of non-preemptive single-machine scheduling problem', *Mathematical programming*, 54 (1992), 353-367.

## 5. TABLES

| Location & Range Parameters for Due Date    | Formulation                                 | Number of Jobs        | Number of Runs where LP relaxation cannot be solved in 1 hr. | Average Number of Nodes              | Number of Test Cases Solved for Optimal Solution[Avg. Computation Time, Seconds] | Number of Test Cases Unsolved in 1 hr. |                                                             | Number of time optimal or best feasible solution obtained compared to other formulation |
|---------------------------------------------|---------------------------------------------|-----------------------|--------------------------------------------------------------|--------------------------------------|----------------------------------------------------------------------------------|----------------------------------------|-------------------------------------------------------------|-----------------------------------------------------------------------------------------|
|                                             |                                             |                       |                                                              |                                      |                                                                                  | Test Cases with no Solution            | Test Cases with Some Integer Solution [Avg. Optimality Gap] |                                                                                         |
| L = {0.5}, R <sub>2</sub> = {0.4, 0.8, 1.4} | Start time & Completion time variables [F1] | 20<br>40<br>60<br>100 | 0<br>0<br>0<br>0                                             | 1627456<br>652137<br>312825<br>25889 | 3 [24.23]<br>0<br>0<br>0                                                         | 0<br>0<br>9<br>9                       | 6 [99.09%]<br>9 [86.51%]<br>0<br>0                          | 3<br>0<br>0<br>0                                                                        |
|                                             | Time index variables [F2]                   | 20<br>40<br>60<br>100 | 0<br>5<br>9<br>9                                             | 2786<br>9<br>0<br>0                  | 4 [1438.43]<br>1 [1239.05]<br>0<br>0                                             | 1<br>8<br>9<br>9                       | 4 [41.44%]<br>0<br>0<br>0                                   | 4<br>1<br>0<br>0                                                                        |
|                                             | Linear ordering variables [F3]              | 20<br>40<br>60<br>100 | 0<br>0<br>1<br>6                                             | 111964<br>1216<br>576<br>63          | 5 [399.2]<br>3 [1437.46]<br>1 [62.45]<br>0                                       | 0<br>6<br>6<br>8                       | 4 [30.74%]<br>0<br>2 [1.26%]<br>1 [1.47%]                   | 8<br>3<br>3<br>1                                                                        |
|                                             | Positional & assignment variables [F4]      | 20<br>40<br>60<br>100 | 0<br>0<br>0<br>0                                             | 1673140<br>712144<br>200655<br>5132  | 6 [784.75]<br>0<br>0<br>0                                                        | 0<br>0<br>0<br>9                       | 3 [44.65%]<br>9 [82.61%]<br>9 [80.92%]<br>0                 | 7<br>6<br>6<br>0                                                                        |

Table 1. Results for Single Machine Total Weighted Tardiness Problem with P = U [1, 100].

| Location & Range Parameters for Due Date    | Formulation                                 | Number of Jobs        | Number of Runs where LP relaxation cannot be solved in 1 hr. | Average Number of Nodes              | Number of Test Cases Solved for Optimal Solution[Avg. Computation Time, Seconds] | Number of Test Cases Unsolved in 1 hr. |                                                             | Number of time optimal or best feasible solution obtained compared to other formulation |
|---------------------------------------------|---------------------------------------------|-----------------------|--------------------------------------------------------------|--------------------------------------|----------------------------------------------------------------------------------|----------------------------------------|-------------------------------------------------------------|-----------------------------------------------------------------------------------------|
|                                             |                                             |                       |                                                              |                                      |                                                                                  | Test Cases with no Solution            | Test Cases with Some Integer Solution [Avg. Optimality Gap] |                                                                                         |
| L = {0.5}, R <sub>2</sub> = {0.4, 0.8, 1.4} | Start time & Completion time variables [F1] | 20<br>40<br>60<br>100 | 0<br>0<br>0<br>0                                             | 2138361<br>585281<br>242333<br>52529 | 0<br>0<br>0<br>0                                                                 | 0<br>0<br>0<br>0                       | 9 [83.99%]<br>9 [92.66%]<br>9 [94.09%]<br>9 [95.57%]        | 0<br>0<br>0<br>3                                                                        |
|                                             | Time index variables [F2]                   | 20<br>40<br>60<br>100 | 0<br>0<br>0<br>0                                             | 2276<br>79185<br>41674<br>8249       | 9 [33.09]<br>5 [38.12]<br>5 [740.35]<br>1 [1260.21]                              | 0<br>0<br>0<br>5                       | 0<br>4 [5.86%]<br>4 [4.19%]<br>3 [21.25%]                   | 9<br>8<br>7<br>4                                                                        |
|                                             | Linear ordering variables [F3]              | 20<br>40<br>60<br>100 | 0<br>0<br>0<br>6                                             | 101249<br>751<br>726<br>24           | 5 [490.17]<br>3 [181.27]<br>0<br>0                                               | 0<br>5<br>7<br>9                       | 4 [50.17%]<br>1 [100.00%]<br>2 [1.43%]<br>0                 | 8<br>3<br>2<br>0                                                                        |
|                                             | Positional & assignment variables [F4]      | 20<br>40<br>60<br>100 | 0<br>0<br>0<br>0                                             | 1777692<br>799491<br>238139<br>8569  | 4 [86.48]<br>0<br>0<br>0                                                         | 0<br>0<br>0<br>2                       | 5 [46.92%]<br>9 [85.70%]<br>9 [91.43%]<br>7 [98.65%]        | 4<br>0<br>0<br>2                                                                        |

Table 2. Results for Single Machine Total Weighted Tardiness Problem with P = U [1, 10].

# **Theoretical Scheduling**

Abstracts

## ON THE COMPLEXITY OF ADJACENT RESOURCE SCHEDULING

C.W. Duin and H.J. van der Sluis

*Operations Research Group*

*Faculty of Economics and Econometrics*

*University of Amsterdam*

Abstract:

We study the complexity of scheduling jobs to adjacent resources. We also show how this problem field, denoted as ARS, is related to known problems in the field of multiprocessor scheduling and strip packing.

As an application of ARS, consider an airport where passengers check in for their flight, joining a waiting line before one or more desks; at the desk the luggage is checked and so forth. For reasons of clarity, management requires that each flight  $f$  is assigned to a number of  $n(f)$  desks, the same ‘interval’ of *adjacent* desks during time interval  $[a_f, a_f+1, \dots, b_f]$  as part of discrete time horizon  $[1, \dots, T]$ . During the

periods  $t$  in  $[a_f, a_f+1, \dots, b_f]$ , the same  $n(f)$  desks (to be chosen) check in customers of flight  $f$  only; in other periods these desks can be assigned to other flights.

The basic ARS problem poses the question: can one find a feasible assignment for all the given flights, if there are no more than  $R$  desks available? A more general problem variant specifies per time period  $t$  a (changeable) number  $n(f, t)$  of required, adjacent desks; here, any flight  $f$  must have in period  $t+1$  as many desks as possible,  $\min\{n(f, t+1), n(f, t)\}$ , in common with previous period  $t$ .

We show that both problems are strongly NP-complete; the general ARS problem is already strongly NP-complete, if the entire planning horizon is restricted to two time periods only ( $T=2$ ). On the other hand, it can be pointed out that the basic ARS problem has a polynomial time approximation scheme, if restricted to some fixed number of time periods  $T_0$ . Further, we determine some cases that are solvable in polynomial time; one of them is a practical case.

Keywords: complexity of scheduling, multiprocessor scheduling, packing

# IMPROVED APPROXIMATION ALGORITHMS FOR BROADCAST SCHEDULING

Nikhil Bansal<sup>1</sup>, Don Coppersmith<sup>1</sup>, and Maxim Sviridenko<sup>1</sup>

<sup>1</sup>IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

## 1. Problem Definition

In this paper we consider two closely related scheduling problems. The first problem is the problem of minimizing the average response time in *on-demand data broadcasting* systems.

The problem of minimizing the response time is formalized as follows: There is a collection of pages  $P = \{p_1, \dots, p_n\}$ . We assume that time is slotted and any page can be broadcast in a *single* time slot. At any time  $t$ , the broadcast server receives  $n_p(t)$  requests for page  $p$  for each  $p \in P$ . We say that a request  $\rho \in \Psi = \{1, \dots, m\}$  for page  $p_{i_\rho} \in P$  that arrives at time  $t_\rho$  is satisfied at time  $c_{i_\rho}(t_\rho)$ , if  $c_{i_\rho}(t_\rho)$  is the first time after  $t_\rho$  when page  $p_{i_\rho}$  is transmitted by the broadcast server. The *response time* of the request  $\rho \in \Psi$  is defined to be the time that elapses from its arrival till the time it is satisfied, i.e.  $c_{i_\rho}(t_\rho) - t_\rho$ . We assume that request  $\rho$  arrives in the end of the time slot  $t_\rho$  and therefore, it cannot be satisfied in the time-slot in which it arrived, i.e. the response time for any request is at least 1. We want to find a broadcast schedule that minimizes the average response time, defined to be  $(\sum_{p \in P} \sum_{t=1}^T n_p(t)(c_p(t) - t)) / (\sum_{p \in P} \sum_{t=1}^T n_p(t))$ .

In the second problem every request has a deadline if a request is not served before its deadline it is lost. The goal is to maximize the total throughput, i.e. the total number of satisfied requests. More formally, every request  $\rho \in \Psi$  has three parameters associated with it. The arrival time  $t_\rho$ , the page  $p_{i_\rho} \in P$  which it requests and the deadline  $d_\rho$ , i.e. if page  $p_\rho$  is not transmitted during the time interval  $(t_\rho, d_\rho]$  the request is lost, otherwise it is satisfied. The goal is to assign one page per time unit during the planning horizon  $[0, d_{max}]$  and maximize the total number of satisfied requests.

## 2. Previous Work and Our results:

We first discuss the average response time problem. Our paper focuses on the offline version of the problem where the request sequence is known in advance to the scheduling algorithm. This problem was shown to be NP-hard by Erlebach and Hall(2002).

Most of the previous algorithmic work on the problem has focused on resource augmentation where the server is given extra speed compared to the optimal algorithm. These results can be viewed as bicriteria approximation algorithms that compare  $k$ -speed approximation algorithm against the performance of an optimal 1-speed algorithm, where a  $k$ -speed algorithm is one that allows a server to broadcast  $k$  pages in each time slot. Kalyanasundaram, Pruhs and Velauthapillai(2000) gave a  $\frac{1}{\alpha}$ -speed,  $\frac{1}{1-2\alpha}$ -approximation algorithm for any fixed  $\alpha$ , for  $0 \leq \alpha \leq 1/3$ . Gandhi, Khuller, Kim and Wan(2002) have given a  $\frac{1}{\alpha}$ -speed,  $\frac{1}{1-\alpha}$ -approximation algorithm for any  $\alpha \in (0, 1/2]$ . Erlebach and Hall(2002) gave a 6-speed 1-approximation algorithm for the problem which was improved to a 4-speed 1-approximation algorithm by (2002). Later, Gandhi, Khuller, Parthasarathy and Srinivasan(2002) give a 2-speed, 2-approximation, (which they improved further in their journal version to a 2-speed, 1-approximation). Very recently, Bansal, Charikar, Khanna and Naor(2005) gave an algorithm that works for arbitrary small speed up factor (such algorithms are referred to as fully-scalable algorithms in the scheduling literature). Their algorithm achieved



an approximation ratio of  $O(1/\epsilon)$  with a  $(1+\epsilon)$ -speed. Here  $(1+\epsilon)$ -speed means that the algorithm is allowed to transmit one extra page every  $1/\epsilon$  time steps.

In the case where no extra speed is allowed, observe that transmitting the pages in the cyclic order  $p_1, \dots, p_n$  repeatedly is a trivial  $O(n)$  approximation. This follows because every request has response time of no more than  $n$  in the cyclic schedule above, whereas it has response time of at least 1 in any optimum schedule. For a while this was essentially the best known guarantee, and only very recently urnal version to a 2-speed, 1-approximation). Very recently, Bansal, Charikar, Khanna and Naor(2005) gave an algorithm that achieves an  $O(\sqrt{n})$  approximation.

We give here an improved algorithm for the problem in the absence of extra speed that achieves  $O(\log^2(T+n))$  approximation, where  $T$  is the time at which the last request arrives. In fact our algorithm achieves a somewhat stronger guarantee in that it gives a schedule with average response time of  $3 \cdot OPT + O(\log^2(T+n))$ , where  $OPT$  denote the value of the optimum solution. We also show that the above algorithm can be modified to yield a slightly better approximation ratio of  $O(\log^2(T+n)/\log \log(T+n))$ .

The minimum average response time problem has also been studied in the online setting. A lower bound of  $\Omega(\sqrt{n})$  without speedup and a lower bound of  $\Omega(1/\epsilon)$  with a speedup factor of  $(1+\epsilon)$ , on the competitive ratio of any randomized online algorithm is known. For deterministic algorithms the lower bound of  $\Omega(n)$  is due to Kalyanasundaram, Pruhs and Velauthapillai(2002). Edmonds and Pruhs(2002) gave a  $(4+\epsilon)$ -speed,  $O(1+1/\epsilon)$ -competitive online algorithm. In their recent paper they showed that a natural algorithm, Longest Wait First, is 6-speed,  $O(1)$ -competitive. Another measure that has been studied in the literature is minimizing the maximum response time (of a request). For this problem, Bartal and Muthukrishnan(2000), gave an  $O(1)$ -competitive algorithm.

The throughput maximization problem was first studied by Bar-noy, Guha, Katz, Naor, Schieber and Shachnai(2002) in a much more general setting. They designed a  $1/2$ -approximation algorithm for the problem. Gandhi, Khuller, Parthasarathy and Srinivasan(2002) designed a  $3/4$ -approximation algorithm for the problem considered in this paper by using a depended randomized rounding. By using the deterministic rounding approach by Ageev and Sviridenko(1999) we obtain a  $5/6$ -approximation algorithm for the throughput maximization version of the broadcast scheduling problem.

# **Timetabling**

Full Papers

# Graphical Tools for Heuristic Visualization

David Ranson and Peter C-H Cheng

Department of Informatics, University of Sussex, Falmer, BN1 9RN, UK

## Abstract

In this paper we present the three visualizations which comprise the VAST examination scheduling system. These graphical interfaces are designed to promote humanization of the scheduling process by supporting human interaction via direct manipulation of the visualizations. The interactive scheduling process is facilitated by visualizations of the unscheduled data, in the Instance view, and the scheduled data, in the Solution view. The third visualization, the Complexity Graph, is used to analyze the relative complexity of examination scheduling problems. A prototype for the visualization of the heuristic processes employed in solving examination scheduling problems is also examined.

## Introduction

This paper introduces the VAST system, a software tool developed as a research aid in the field of examination scheduling. VAST contains a set of visualizations designed to be used for analyzing the complexity of examination scheduling problems, helping to solve these problems and visualizing the heuristic strategies employed in solving them.

Before discussing the motivations behind building VAST it is useful to look at the reasons for investigating this topic, present the university examination scheduling problem, review VASTs' direct predecessors and also to place the system into the broader research context.

The examination scheduling problem is interesting as the subject of these interactive visualizations as the basic structure of the examination scheduling problem can be clearly defined; however when scaled up to real life situations the problems become highly complex making valid solutions difficult to find. It is hoped that the diagrammatic tools presented here can be used to help successfully reduce this complexity.

Fundamentally the examination scheduling problem involves assigning resources, rooms and times, to exam events whilst ensuring that the number of violations, such as clashes where a student is scheduled to one or more exams simultaneously, are minimized. Other violations considered here are: room and time overflows, where exams are scheduled in locations without the required capacity or duration; and, consecutive violations, where a student has one or more exams in immediate succession. Throughout this discussion 'cluster' refers to a set of exams that do not share any common students (conflict) and can therefore all be scheduled in the same timeslot without causing any clash violations.

The STARK[1] diagram for examination scheduling is a visualization supporting examination scheduling. This visualization allows users to interactively refine schedules by directly manipulating exam icons. The STARK diagram is used within the HuSSH system which adds functionality to select and apply automated heuristics to schedule and modify solutions. HuSSH has been used to discover the individual and varying heuristics humans use when scheduling exams and experiments have demonstrated that novice users are capable of improving automated solutions [2]. A variant of this STARK diagram is used for the solution visualization in the VAST system and other features, such as the visual exam and constraint representations, used in VAST were first found here.

The Examualiser programs [3, 4] were visual tools previously created by the author and others as undergraduate projects. The initial visualization was focussed on representing the complexity of exam problems[3]. The second of these projects dealt with creating structured representations for exams before scheduling occurred[4]. The advantages of this approach are also exploited in VAST.

There are a number of commercial systems available for examination scheduling, such as Syllabus Plus[5] or CelCat[6], each of which use propriety user interfaces which often appear "to be almost an afterthought"[7]. These systems are primarily aimed at automated scheduling offering varying degrees of control to the users. The VAST interfaces however are designed to allow the users complete control throughout all stages of the scheduling process. This work is part of wider research studying different visualizations that can be used in representing scheduling processes and the problem solving heuristics (human or computer) used. One goal of this research is to design visualizations that will allow the structure and processes of heuristics to be made visible to users. This may increase the user's confidence in the heuristic results and also allow users to remain actively involved with the scheduling

process at all stages, especially when the visualizations are interactive. An example of a visualization that goes some way to achieving this goal on a limited group of heuristics will be given towards the end of this paper.

The approach taken by this paper will be to present each of the components of VAST, including the rationale and advantages for each. These elements are the Complexity Graph, the Instance Visualization and the Solution Visualization. This will lead to a discussion on how the system fits together as a whole to allow interactive scheduling and illustrating how heuristic processes can be made visible. The first section of this paper will introduce the Complexity Graph, explaining the need for such a diagram, before going on to look at how the diagram can be used throughout the scheduling process. Secondly the Instance and Solution visualizations will be presented demonstrating how they are used in the scheduling process and how the manipulation of these diagrammatic tools can simplify both the scheduling process and the users understanding of the scheduling process. Finally the paper will show how these pieces fit together and then end with a concluding discussion.

## Visualizing Complexity

When working on different types of problems one question that is invariably posed will be “How hard is this problem to solve?” This may be in relation to other similar problems, or instances of the same problem. Mathematics helps answer the first query by providing a means of classifying different problems, for example the examination scheduling problem is known to be NP-Hard[8]. In contrast the visualization discussed here attempts to provide a visual solution to the second question: how hard are examination scheduling problems to solve relative to each other? Further how do different parts of the same problem, for example different departments, relate to each other in terms of difficulty to schedule?

There are three reasons for using this tool to study the structure and complexity of an examination scheduling problem. Firstly not only does the diagram provide an insight into how hard an instance is to solve but it also provides a basis for comparing instances to each other.

A second reason for using this visualization for analyzing the complexity of an examination scheduling problem instance is to see which parts of the problem look like they are going to be the hardest to solve. Many heuristics involve trying to solve the hardest parts of problems first, for example by initially scheduling exams with the highest number of clashes. Although this is relatively easy for an automated system it is not so easy for a human scheduler, especially when the criteria for selecting exams becomes more complicated.

The third reason given for using this visual device is to see how much of a problem has already been solved. As will be shown the complexity graph presented here is not just a static diagram but is an essential and interactive part of the scheduling process supported by the VAST system.

Previously explored visualizations[3] attempted to represent the complexity of a scheduling problem using clusters. This solution was to cluster the exams and then display the number and magnitude of these clusters. However it is questionable whether the number of clusters is a good indicator of problem complexity and, further, the number of clusters is not exact but dependent on the heuristic used to generate them.

Another feature of the problem, which does appear to be a good indicator of the difficulty of solving an examination scheduling problem[9], is the number of potential clashes (or exam intersections) that exist. This value is not dependent on any algorithm, and is used here as a measurement of the problems complexity. The approach adopted by the VAST complexity graph is to visualize the number of potential clashes along with the overall size of the problem, the number of exams, departments and year groups. By making these components distinct it is hoped that the location of complexities and potential scheduling difficulties in the problem instance will be made more apparent.

Figure 1c illustrates a department in Complexity Graph visualization. The square boundary represents the department; its area indicates the number of exams in the department so comparing the size of two departments involves simply checking the relative areas. Figures 1a, 1b and 1c all show the same department, scaled down in the first two cases. Inside the department each of the circular points, arranged in a spiral to minimise overlapping lines, represents a single exam as shown in figure 1a.

The second square, figure 1b, shows the highlighting function where exams with a number of potential clashes over a threshold value are highlighted by a red circle. The user can easily change this threshold value to determine which exam(s) in a department are causing the greatest number of clashes and contributing to its complexity.

If two exams intersect, a potential clash, then a line is drawn between these two exams; this is illustrated by figure 1c. The width of each line is proportional to the number of students shared between the exams and the colour identifies the year the shared student belongs to. A legend is provided which identifies each year group, allows them to be individually hidden as required, and also allows the highlighting threshold to be changed.

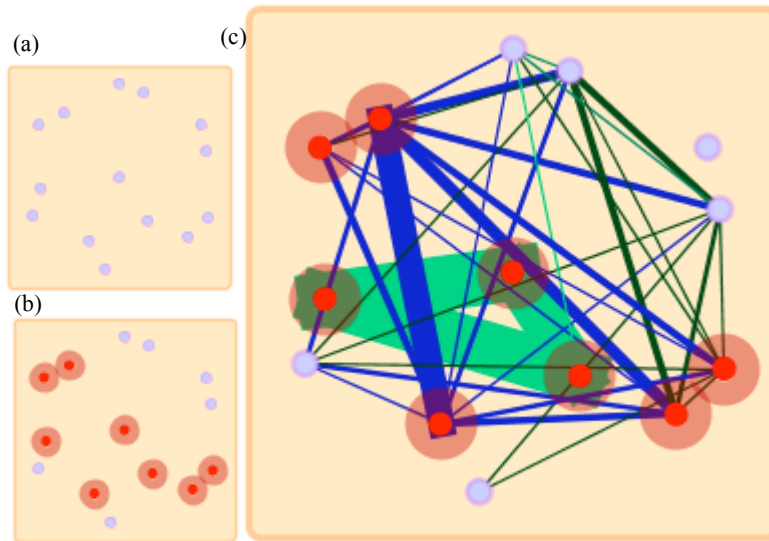


*Figure 1.* Three perspectives of a single department in the Complexity Graph:  
 (a) A single department showing the exam nodes.  
 (b) The same department highlighting exams which clash with a number of other exams.  
 (c) The same department showing the clashes between exams. The optional line thickness is turned on to indicate the number of students involved in the clash which can help to distinguish clashes involving the same students.

Situations where departments have a higher number of conflicts generally need to be given priority in the scheduling process, and are assumed to be harder to schedule than departments with much fewer conflicts. Here this is represented by the number of lines in the department becoming more indistinguishable the greater the number of conflicts; that is scheduling complexity is represented by visual complexity. The optional line thickness, indicating the number of students involved in each clash and turned on here, is largely irrelevant although it can be used to distinguish groups of exams where the same students are causing all the conflicts. On its own a single department imparts relatively little information that can be used to create an overall scheduling strategy, however when a complete complexity graph is inspected it can suggest ways to move forward in the scheduling process.

Figure 2 is a complete Complexity Graph showing all the potential clashes that can exist between exams within the different departments. By inspection it is possible to distinguish departments that are distinctly sparser than others, and departments that are smaller than others; this suggests that these will be easier to schedule than the larger, more daunting looking departments. A good use of this information would be to start the scheduling process by scheduling exams from the departments perceived as more difficult to ‘solve’ and then moving on to the departments with fewer constraints.

Analyzing the individual departments further provides even lower level pointers on how to start scheduling. In figure 1c there is a solitary exam with no constraints, in the upper right quadrant, the implication is that this exam can easily be left to very late on in the scheduling process without expecting it to cause many problems.

It is a simple task to identify groups of exams which when removed can significantly improve the difficulties of scheduling a department. In Figure 1c the same coloured line configurations, such as the large triangle, are examples of such groups. These different coloured line configurations represent groups of exams taken by a particular year group. It is common to find in these visualizations that the first year component of a department causes a significant proportion of the overall number of potential clashes. Scheduling these exams first might be a good heuristic to reduce the complexity of the problem; again this use of the visualization suggests another order for scheduling the exams. A second mode of the Complexity Graph shows potential clashes between exams in different departments, a feature much more common nowadays with the widespread occurrence of modular courses. As illustrated by the partial snapshot in figure 3 this form of the diagram appears more complex than the individual departments view but can be used to visually perceive the complexity of the overall problem.

When two exams are scheduled all potential clashes between these exams are removed from the diagram. This enables the scheduler to see how much of a problem has already been solved; this reduction is shown in figure 4.

This is an improvement on just counting how many exams out of the total have already been scheduled as this alone does not take into account the relative contributions each exam makes to the overall complexity. By design the complexity graph does not show clashes between are forced to clash in the solution; these are visible in the solution visualization.

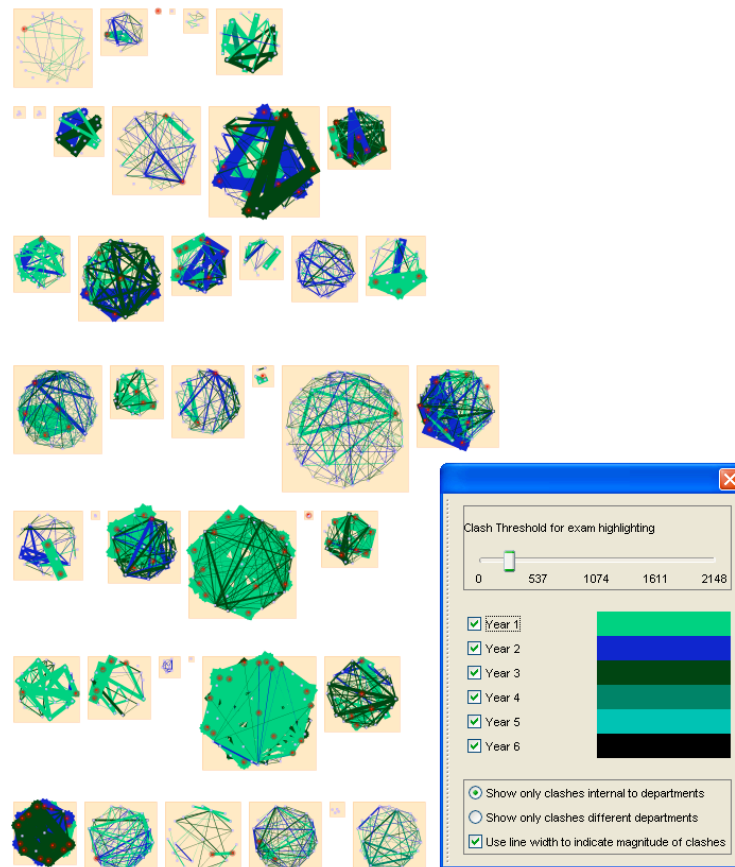


Figure 2. A Complete Complexity Graph including the Legend

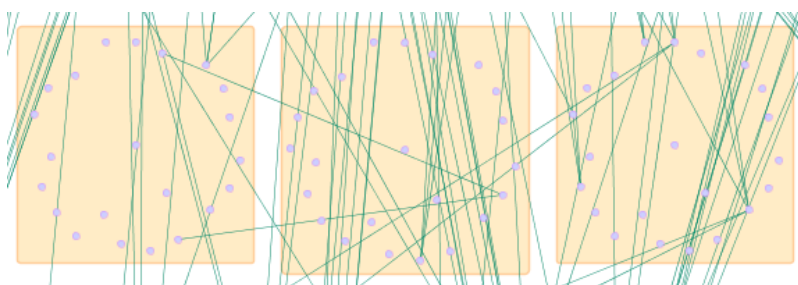


Figure 3. Part of the graph showing potential clashes between Departments

By watching how the complexity graph changes and simplifies during the scheduling process it is possible to determine how far the problem has been solved. If no lines are to be found at all in the complexity graph the entire problem (in respect to clashes) has been solved. This can be used in conjunction with a traditional evaluation function as it gives an indication of not how good a current solution is, but how much more needs to be done to reach an optimal solution. An advantage over traditional evaluation functions is that the complexity graph can also identify parts of the problem, such as departments or individual exams, which are causing the greatest difficulties in solving the problem.

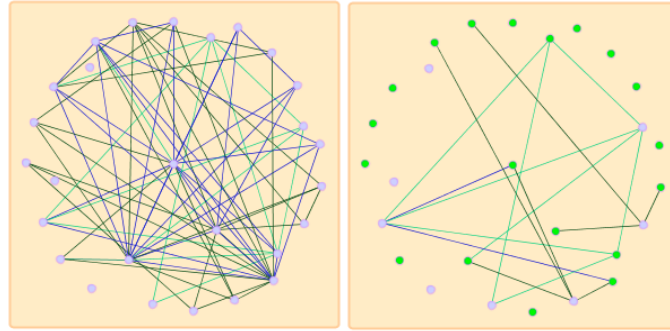


Figure 4. The same department before and after some exams have been scheduled.

## The Visual Scheduling Tools

The visualization tools in the VAST system provide a consistent interpretation of the data making up an examination scheduling problem instance. While the individual visualizations, including the more abstract Complexity Graph, are designed for specific purposes they should not be viewed only as separate tools but as complementary parts of the system.

The two visualizations described in this section, the Instance view and the Solution view, deal with the view of unscheduled and scheduled examination data respectively. The interactive scheduling process supported by the VAST system makes use of both these visualizations with the user selecting *Strings* of exams from the Instance visualization and then scheduling these exams in the Solution visualization. This process will be explained in the coming sections.

Each of the scheduling visualizations provides a view of the exams found in the instance and the constraint relationships between those exams. In order to do this they share the concept of an exam glyph, an icon that represents the exam and its attributes in both of these diagrams. An exam glyph takes the form of a rectangle with the dimensions encoding the number of students taking the exam, as the height of the glyph, and the duration of the exam, as the glyphs' width[1]. The colour of the glyph is used to encode the group to which the exam belongs; this will normally be either a department group or cluster group and is controlled by the user to aid the task they are currently performing.

### Instance Visualization

The VAST Instance Visualization represents all the unscheduled examination data found in an examination scheduling instance. This unscheduled data is initially ordered arbitrarily and unstructured, there are no implicit hints found as to a consistent structure for displaying this information. The Instance visualization uses all the available data in order to create suitable structures, in this way transforming the lower level data, lists specifying the problem, into higher-level visual structures which actually aid the scheduling process.

Figure 5 is a snapshot of the default instance visualization for an instance dataset. Here the exam glyphs are arranged into columns dependent on the cluster to which the exam belongs, within each column exams are ordered based on the number of students. There is no ordering from left to right of the cluster columns; any perceived ordering is simply an artefact of the greedy clustering algorithm used.

As with all the visualizations users can zoom in and out, view in full screen scroll across the visualization, making even extremely large datasets visible whilst still allowing users to focus in on individual features. Each exam is clearly represented by its own glyph, with the attributes encoded by the dimensions making for easy visual comparison between the exams. The lower right window show how the exams are currently organized and coloured, the user can also change these settings here.

The real power of the Instance visualization as an aid to scheduling, rather than it being just another (albeit visual) list of exams, lies in the visual organisation of the exams facilitated by the graphic. Exam *set viewing* allows the user to limit visible exams to a particular set. In Figure 5 the exams are viewed globally so that all the exams are visible but exams can also be viewed in department sets so that only a single department is visible at any time.

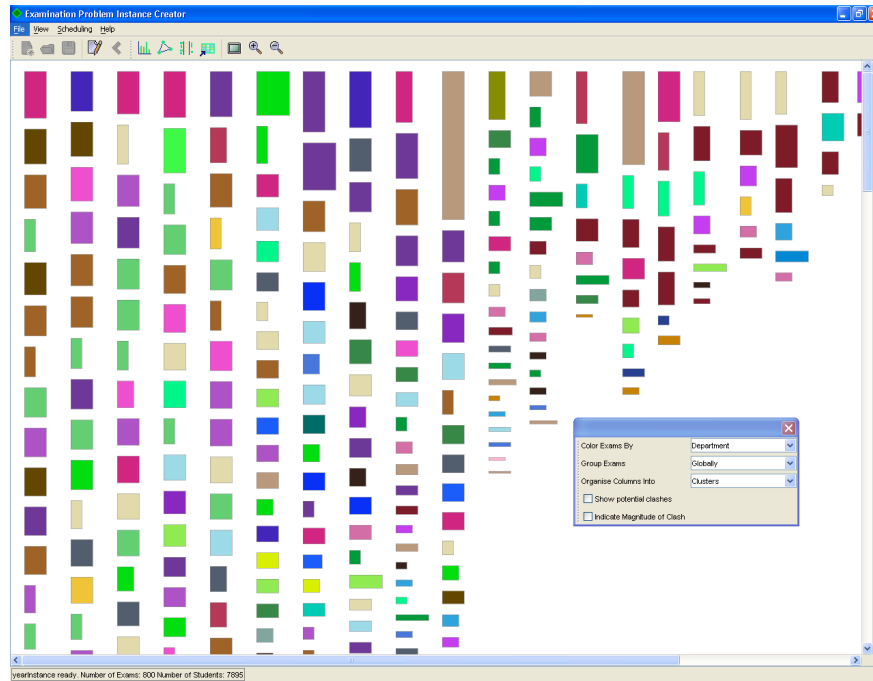


Figure 5. The VAST Instance Visualization.

In addition to limiting the visible exams the Instance view also enables exams to be arranged into organizations structured to make it easier for users to make scheduling selections. When organized into clusters, as in figure 5, entire columns of exams can be scheduled in a single timeslot with the knowledge that no clash or consecutive violations will occur. The exams can also be coloured in order to highlight groups of exams that might belong together and be scheduled at the same time. Alternatively exams could be organized into columns by the duration. In contrast to the previous graphic, figure 6 illustrates an Instance View with the visible exams limited to a single department, organized by Duration and coloured in respect to the cluster they belong to. Also displayed are the potential clashes that will occur if some exams from different clusters are scheduled at the same time. Note that no clash lines exist between any exams in the same cluster.

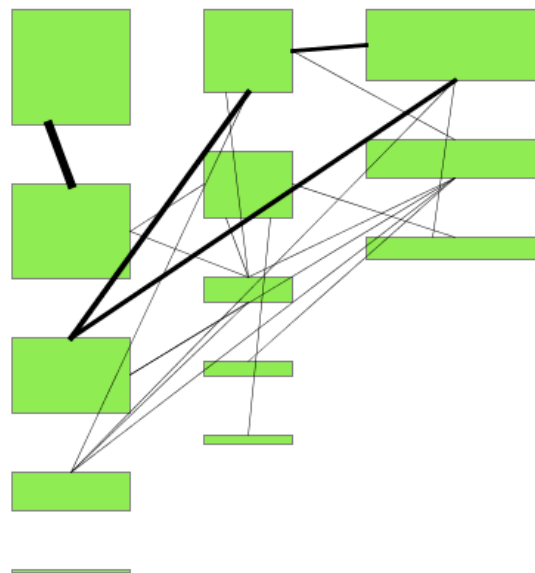


Figure 6. An Instance view of a department showing potential clashes between exams.



## Solution Visualization

The last visual component of the VAST system represents the solution to an examination scheduling problem as it is being solved. It provides an interactive visualization showing both where exams have been scheduled and also the location of any constraint violations that may occur. The visualization is essentially a simplified implementation of the STARK examination scheduling diagram[1]. The diagram is constructed as a table like scheme with timeslots along the horizontal axis and rooms along the vertical. The names of the rooms and times are replaced with numbers and letters respectively; this emphasizes the ordering of the timeslots, whilst abstracting away from the arbitrary physical names of the rooms.

Each timeslot has a width, which represents the duration, and each room has a height representing the capacity of that room. As this encoding is the same as for the exams it has the advantage that it explicitly forces exam glyphs to be placed in locations where they physically fit, meaning exams will be scheduled only into timeslots with sufficient duration and capacity. Figure 7 shows part of a solution, in the visible area there are a number of consecutive violations shown by black lines linking the exams involved. Any clash violations would be represented on the diagram by red lines between the affected exams. Timeslots are grouped into days, so that overnight or weekend breaks are shown by gaps in the background box, as indicated by 'A'; consecutive violations do not occur across overnight gaps. Once again the exam glyphs can be colour coded based on the departments or clusters they belong to.

## The Visual Scheduling Process

Whilst all these diagrams are useful for analyzing problems they also work together to enable interactive scheduling. This process is made up of two steps: Exam Selection and Exam Scheduling.

There are two reasons for breaking down the scheduling process into these components; primarily it minimizes the need to switch between the Instance and Solution views but it is also very flexible and the methodology can be used to model many human and automated scheduling heuristics. All the heuristics considered here are examples of Sequential methods[10]; however this approach may also be suitable for other classes of heuristic such as Cluster methods.

The first stage, Exam Selection, involves selecting exams based on some criteria such as the number of clashes. Another heuristic, sometimes used by humans, is to select the exams with the largest durations or capacities first and then moving on to the smaller ones. In VAST these exams are selected in Strings, clicking on an exam will add it to a string clicking it again will remove it from the String.

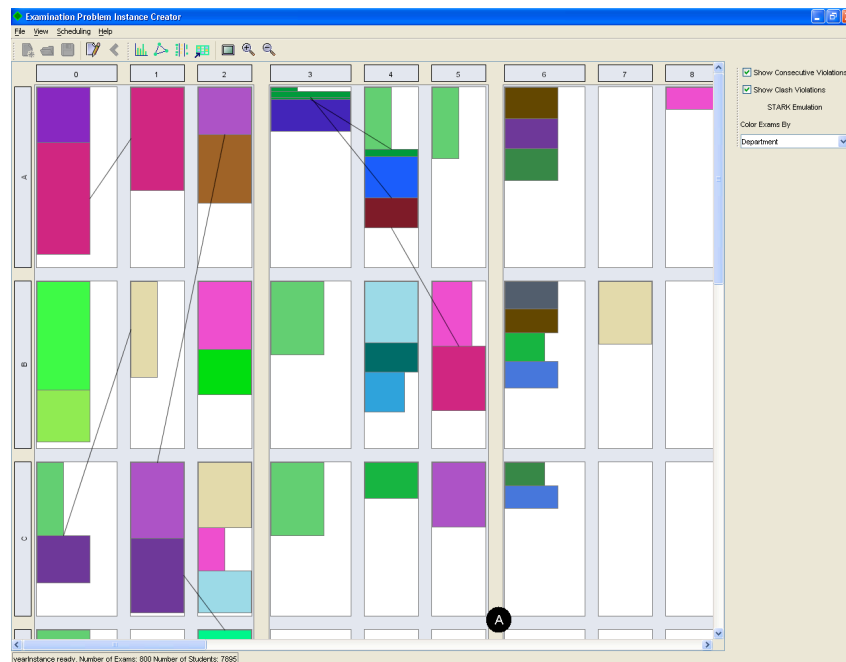


Figure 7. The VAST Solution Visualization

Figure 8 illustrates a String of exams selected from the Instance Visualization, and a String selected in the complexity graph. Note that the first six exams selected in the complexity graph were highlighted ones indicating they contribute a greater number of potential clashes. This selection can be performed manually, using the Complexity Graph and Instance views to select Strings based on any decided criteria, or could be automated.

The second stage, Exam Scheduling, involves assigning Exams on the String to locations in the Solution visualization. The first exam to be scheduled is always the first exam added to the String, after this the exams are scheduled in the order they were added to the selection String. The next exam to be scheduled is always displayed, as illustrated by Figure 9.

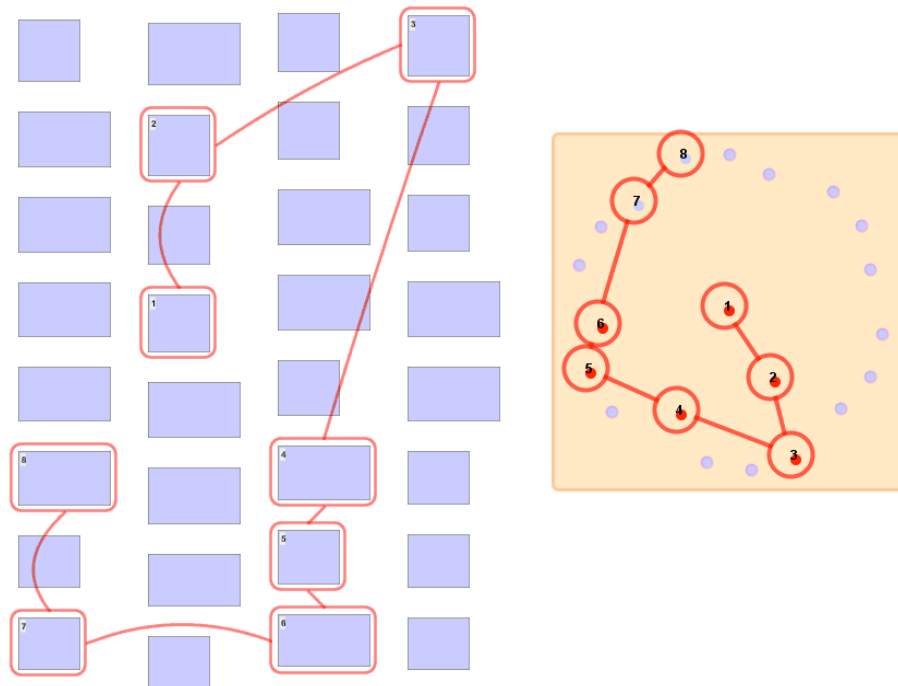


Figure 8. A String of Exams shown in the Instance and Complexity graph Visualizations

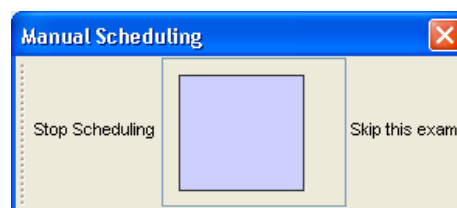


Figure 9. The Manual Scheduling Window showing the next exam found on the String

After an exam has been scheduled the Exam Glyph in the Instance visualization is replaced with an outline of the glyph, this allows users to tell at a glance which exams have already been scheduled, the location and time to which the exam has been scheduled are also added to the glyph as illustrated by figure 10.

Again it is also possible to automate this scheduling process, and a programming interface (API) exists for creating VAST scheduling algorithms. The usefulness of using VAST with automated heuristics is that the heuristic process is made visible to the user. By following the selection strings, displayed on the Complexity Graph or Instance view, it is possible to see the order in which the exams were selected and scheduled.

This is one possible way of explaining heuristics to users and giving them a greater understanding of how the heuristic they are using actually works. It also allows different heuristics to be compared visually; comparing the

two selection strings will make visible the exact points at which the heuristics make different selection decisions and the paths diverge. These prototype heuristic visualizations clearly demonstrate the potential of using this approach.

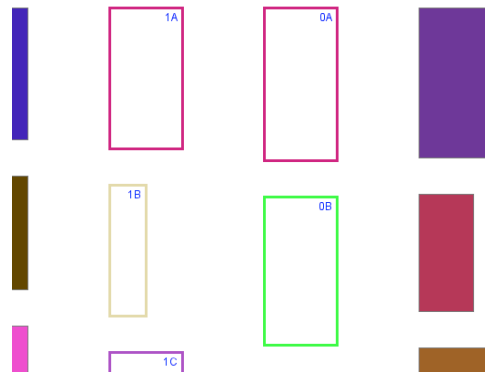


Figure 10. Scheduled Exams as they appear in the Instance Visualisation

## Discussion

The system presented in this paper was developed to fit a very narrow set of specific requirements suited to the authors' needs, however the principles used are adaptable to a wider range of timetabling and related problems.

Whilst not being a complete scheduling system the tools discussed here can be used to simplify the scheduling process and provide interactive elements currently missing from many automated scheduling systems. When combined the visualizations aim to increase the users' knowledge of how the scheduling process is being performed, keeping them involved at all stages of the process.

This humanization of the scheduling process poses a number of advantages over existing fully automated systems. Users maybe more involved in the interactive scheduling environment, and hopefully feel in more control when they can see what is happening and perform scheduling interactively. As the understanding of the process develops so may the users' confidence in existing heuristics; users may even be given confidence and inspiration to experiment with, and create new heuristics.

Evaluations of the VAST system are planned. Visualizations will also be explored into other parts of the scheduling process, such as the clustering process, and more advanced heuristic visualizations and tools for other classes of heuristics in examination scheduling problems.

## Acknowledgements

This work would have been impossible without the previous work and suggestions from Dr Samad Ahmadi.

## References

1. Cheng, P.C.-H., R. Barone, P.I. Cowling, and S. Ahmadi, *Opening the Information Bottleneck in Complex Scheduling Problems with a Novel Representation: STARK Diagrams*, in *Diagrammatic representations and inference: Second International Conference*. 2002, Springer-Verlag: Berlin. p. 264 - 278.
2. Ahmadi, S., R. Barone, E.K. Burke, P. Cheng, P. Cowling, and B. McCollum. *Integrating human abilities and automated systems for timetabling: A competition using STARK and HuSSH representations at the PATAT 2002 conference*. in *Practice and Theory of Automated Timetabling*. 2002. KaHo St.-Lieven, Gent, Department of Industrial Engineering, Belgium.
3. Fullbrook, M., N.J. Sim, L.S. Dean, D.J. Ranson, C.F. Duncan, and R. Jobanputra, *Visualisation of the data for examination timetabling problem*. 2001-2002, Unpublished undergraduate project report, University of Nottingham.
4. Ranson, D., *Timetable constraint evaluation system*. 2003, Unpublished undergraduate dissertation, University of Nottingham.
5. Syllabus Plus Webpage, 2004, retrieved on January 19, 2005 from <http://www.scientia.com/products/>

6. CelCat Webpage, 2005, retrieved on January 19, 2005 from <http://www.celcat.com/>
7. McCollum, B., S. Ahmadi, E.K. Burke, R. Barone, P.C.-H. Cheng, and P. Cowling. *A Review of Existing Interfaces of Automated Examination and Lecture Scheduling Systems(Abstract)*. in *Practice and Theory of Automated Timetabling*. 2002. KaHo St.-Lieven, Gent, Department of Industrial Engineering, Belgium.
8. Fisher, J.G. and R.R. Shier, *A Heuristic Procedure for Large-Scale examination scheduling problems*. Congressus Numerantium, 1983. **39**: p. 399-409.
9. Burke, E.K., A.J. Eckersley, B. McCollum, S. Petrovic, and R. Qiu. *Analysing Similarity in Examination Timetabling*. in *5th International Conference on the Practice and Theory of Automated Timetabling*. 2004. Pittsburgh, PA USA.
10. Burke, E.K. and S. Petrovic, *Recent Research Directions in Automated Timetabling*. European Journal of Operational Research, 2002. **140**(2): p. 266-280.

# **Timetabling**

Abstracts

# COURSE TIMETABLING

*Extended tiles for university timetabling*

Mieke Adriaen, Patrick De Causmaecker and Greet Vanden Berghe

*KaHo Sint-Lieven*

*Department Information Technology*

*Gebr. Desmetstraat 1*

*9000 Gent*

*Belgium*

*{Mieke.Adriaen,Patrick.DeCausmaecker, Greet.VandenBerghe}@kahosl.be*

## Abstract

Key words: Modelling, Timetabling

Titan [1] (Transparent Interactive Timetabling based on Agents Negotiation) is a university timetabling application that mimicks the steps taken when manually constructing timetables. The system defines *sessions* as basic building blocks for the timetables. A session is characterised by a date, time, duration, teacher group, student group, subject and location.

Investigating the working methods of human timetablers showed that we can split our program in three basic blocks: the construction of timetable elements, a grouping method and scheduling the generated groups. We also noticed that human timetablers prefer interactive software rather than black box solutions.

In the first step of the algorithm, we generate the *timetable elements* that are necessary to meet the student programs. Timetable elements are sessions for which time and location are not yet specified. They are generated for a selected period such as a semester.

The next step in the process is based on the approach for solving high school timetabling as presented in [2]. In this algorithm the meetings are grouped into new units called *tiles*.

The problem described in [2] is less complex compared to the university timetabling problem. In Kingston's paper time is divided into periods of equal length and timetables are constructed on a weekly or fortnightly cycle.

We generalise the tiles concept to incorporate the extra degree of freedom that is applied in typical university systems. Instead of tiles living in a two dimensional world, we use *pillars* in a three dimensional structure. Pillar construction is done manually by users or can be automated. As in [2] this leads to a dramatical reduction in complexity. Pillars allow to capture user preferences as well as experience and historical information while meeting a large number of constraints automatically.

In the third step pillars are manipulated using a meta-heuristic based approach.

This model was implemented and tested against historical data sets. Results will be discussed.

## REFERENCES

- [1] M. Adriaen, P. De Causmaecker, G. Vanden Berghe: *Decentralised Course Timetabling in a Large Hierarchical Organisation*, Proceedings of The 1<sup>st</sup> Multidisciplinary International Conference on Scheduling: Theory and Applications, Nottingham, 2003, 60-65.
- [2] J. H. Kingston: *A Tiling Algorithm for High School Timetabling*, Proceedings of the 5<sup>th</sup> International Conference on Practice and Theory of Automated Timetabling, Pittsburgh, 2004, 233-249.

# A predictive-reactive approach for RCPSP with labour constraints and time-dependent activities requirements

LE. Drezet <sup>\*,+</sup> and JC. Billaut <sup>\*</sup>

<sup>\*</sup> *Université François-Rabelais de Tours,  
Laboratoire d'Informatique, 64 avenue Jean Portalis,  
37200 Tours, France*

`laure-emmanuelle.drezet@etu.univ-tours.fr`

<sup>+</sup> Eskape

37100 Tours, France

## Abstract

We consider a project scheduling problem where resources are employees and activities requirements are time-dependent. Employees have different skills and legal constraints have to be respected. We propose a predictive approach solving the scheduling problem and building the employees timetables. Then a reactive approach is proposed, to react in real time to some unexpected events.

**Keywords:** *Project scheduling, Workforce management, predictive-reactive approach.*

## 1 Problem description

We consider in this paper a particular Resource Constrained Project Scheduling Problem (RCPSP), where the resources needed to perform the activities are employees. By considering human resources, we have to take specific constraints into account due to the skills of the employees and due to workforce legislation. These constraints are called legal constraints.

A project is composed of activities which are subject to precedence constraints and which require a given amount of resource to be performed. The resources needed to perform the activities have a limited capacity. The Resource Constrained Project Scheduling (RCPSP) consists in scheduling the activities so as to minimise the project makespan. Many approaches and efficient lower bounds have been developed to solve this NP-Hard problem (See [1] for more details). But to the best of our knowledge, few approaches consider the specificities of human resources.

In our problem, an activity is characterized by a duration, a release time and a due date. The resource requirements are time-dependent and are bounded between a minimum and a maximum time-dependent value. Hence, two resource profiles are associated to each activity. Another constraint imposes that among all the employees present at a same time, a given



number have to masters some particular skills. These constraints are due to maintenance contracts, where one client can force the organization to always have some present employees able to solve specific problems. We call these constraints skills requirements.

The problem comes from an industrial need, where a project corresponds to a software product development. State-of-the-art surveys on workforce scheduling and RCPSP are presented. To the best of our knowledge, this problem has never been tackled in the literature before. The approach we propose is decomposed into two phases. The first phase is the predictive phase, where the RCPSP is solved by using heuristic algorithms. The second phase is a reactive phase, where the solution is modified in order to take some unexpected events into account, without having to solve the whole problem again.

## 2 Predictive approach

We suppose in our problem that the employees' days-off have already been set, that the company works only five days a week and that the planning horizon is comprised between two and four weeks. The objective of the problem is to minimise the maximum lateness of the project. To solve the problem we define some greedy algorithms based on priority-rules. These algorithms minimize the maximum lateness and try to minimize the number of violated legal constraints and unsatisfied skills requirements. The best solution of these algorithms is used as an initial solution of a Tabu search algorithm, which increases both the quality and the feasibility. Computational experiments are conducted and results are presented.

## 3 Reactive approach

The horizon of the reactive phase is one day. The forecasted plan can become unfeasible due to the increase of the duration of one activity, to the unavailability of one or several resources or to unexpected activities. The problem is to recover the solution feasibility and to minimize the number of modified timetables. We propose a Tabu search algorithm to solve this problem. Computational experiments are conducted and results are presented.

## References

- [1] Demeulemeester E. and W. Herroelen, Project Scheduling : A Research Handbook, *International Series in Operations Research and Management Science, Volume 49*, Kluwer, 2001.

# Solving a Timetabling Problem using Benders' Decomposition

Subhash C. Sarin<sup>\*</sup>, Yuqiang Wang, Amrisha Varadarajan

*Grado Department of Industrial and Systems Engineering,  
Virginia Tech, Blacksburg, Virginia, 24061*

## Abstract

The university timetabling problem comprises of scheduling a certain number of meetings between faculty and students over a time period such that resource constraints, faculty preferences, administrative and course requirements are satisfied. The large volume of data coupled with the idiosyncratic constraints makes this a formidable problem to solve optimally. In a typical American university, the timetabling problem involves more than two thousand classes, over a hundred classrooms and several hundred teaching personnel.

At Virginia Tech, classes presently are scheduled one semester in advance. The university registrar's office commences the scheduling process by requesting course profiles from each department. The course profile includes among other details, the number of credits, the capacity of the course, the faculty's name, his/her preferred timeslot and three candidate classroom buildings with differing levels of priority. After receiving all the profiles, the registrar's office feeds the data into a software package, that attempts to determine the best possible classroom for each course in one of its three preferred buildings at its preferred timeslot while satisfying the following two strong constraints: (1) No classroom can hold more than one class at any time; (2) No faculty can teach more than one class at any time. Since the number of available classrooms is limited for each building and the software is not designed to manipulate the course characteristics, occasionally the software may fail to generate a "feasible" room for some courses. When this happens, the staff at the registrar's office manually schedules these "infeasible" courses by requesting the department to alter the course characteristics. Clearly, this scheduling mechanism is neither optimal, nor flexible. The need for a superior strategy to optimally allocate university resources over time in a manner that satisfies all hard and soft requirements has been the key motivation for our study.

The timetabling problem and in particular the university timetabling problem has been tackled using various approaches namely integer programming [Daskalaki, 2004&2005], knowledge

---

Corresponding author. Email: sarins@vt.edu.

based scheduling within an object oriented paradigm [Gunadhi et al, 1996], genetic algorithms [Ueda et al, 2004], integer goal programming (IGP) with genetic algorithms [Mirrazavi et al, 2003] and constraint handling rules [Abdennadher and Marte, 2000]. Daskalaki et al. [2004] apply their IP formulation to the fall semester timetabling problem of an engineering department at a university and present results for a maximum problem size of 119 courses to be accommodated in 18 rooms with 326 teaching periods. In a later study [2005] they propose a two stage approach, where the first stage relaxes the computationally heavy constraints and the second stage recovers these constraints by solving a day-to-day timetabling problem for a locally optimal solution. Both stages are solved using an IP formulation that employs a traditional IP solver and results presented for the same test problems undertaken in their previous study [2004] show a reduction in computation times without a significant loss in solution quality. Gunadhi et al. [1996] create an automated timetabler using an object oriented paradigm to generate schedules for 296 classes, 110 instructors and 1796 students to be accommodated in 41 rooms. Ueda et al. [2004] compare various genetic algorithms for timetabling problems generated using an automatic timetabling problem generator for Japanese universities with characteristics that differ significantly from their western counterparts. Here timetables are generated before students decide on the courses. Mirrazavi et al. [2003] discuss a two phase approach where the first phase allocates lectures to rooms and the second allocates start-times to lectures. Both the phases are modeled as an IGP but the second phase is solved using a genetic algorithm whereas the first phase uses a commercial solver. The concept is applied to a science department at a university with a problem size of 212 courses spanning over 436 lectures and 351 student groups to be accommodated in 33 rooms. Abdennadher and Marte [2000] use constraint handling rules for timetabling courses offered at a science department in a university. Parts of previous year timetables are reused to reduce computational time and their study reports on scheduling 89 courses within 42 time periods in about 5 minutes.

Most of the studies pertaining to university timetabling have focused on applications pertaining to course scheduling at a single or at most few departments in the university. In contrast we undertake the timetabling problem for all departments in the university and our problem encompasses thousands of classes to be accommodated in hundreds of classrooms spread over tens of buildings. This problem size is relatively larger compared to the problems encountered in previous studies described above while retaining a similar level of complexity in terms of requirements to be satisfied. In our work, we begin by first describing precisely the problem to be solved followed by an insight that

permits the problem to be decomposed into two subproblems. The first subproblem deals with scheduling courses for Monday, Wednesday and Friday (MWF) whereas the second subproblem deals with course offerings for Tuesday and Thursday (TTh). Each of these subproblems is formulated as an integer linear programming problem in 0-1 variables with the objective of minimizing the total distance traveled by the faculty from their offices to the classrooms, where their classes are assigned. The special structure of these subproblems combined with the monotonic nature of the objective function permits an efficient reformulation of the integer program as a mixed integer linear program (MIP) that still guarantees integer optimal solutions. We then present a solution strategy, which employs decomposition through the Benders' partitioning procedure for mixed integer programs for solving the subproblems. And finally, we conclude with a report on relevant computational results.

#### **Bibliography:**

- [1] S. Abdennadher and M. Marte, "University course timetabling using constraint handling rules," *Applied Artificial Intelligence*, vol. 14, pp. 311-325, 2000.
- [2] S. Daskalaki and T. Birbas, "Efficient solutions for a university timetabling problem through integer programming," *European Journal of Operational Research*, vol. 160, pp. 106-120, 2005.
- [3] S. Daskalaki, T. Birbas, and E. Housos, "An integer programming formulation for a case study in university timetabling," *European Journal of Operational Research*, vol. 153, pp. 17-135, 2004.
- [4] H. Gunadhi, V. J. Anand, and Y. W. Yong, "Automated Timetabling using an Object-Oriented Scheduler," *Expert Systems With Applications*, vol. 10, No. 2, pp. 243-256, 1996.
- [5] S. K. Mirrazavi, S. J. Mardle, and M. Tamiz, "A two-phase multiple objective approach to university timetabling utilising optimisation and evolutionary solution methodologies," *Journal of Operational Research Society*, vol. 54, pp. 1155-1166, 2003.
- [6] H. Ueda, D. Ouchi, K. Takahashi, and T. Miyahara, "Comparisons of genetic algorithms for timetabling problems," *Systems and Computers in Japan*, vol. 35, No. 7, pp. 691-701, 2004.

# **Transport Scheduling**

Full Papers

# APPLYING THE EXTENDED NETWORK SIMPLEX ALGORITHM TO DYNAMIC AUTOMATED GUIDED VEHICLE SCHEDULING

Hassan Rashidi  
*Department of Computer Science*  
*University of Essex, Colchester CO4 3SQ,*  
*U.K.*  
*Email: [hrashi@essex.ac.uk](mailto:hrashi@essex.ac.uk)*

Edward P. K. Tsang  
*Department of Computer Science*  
*University of Essex, Colchester CO4 3SQ,*  
*U.K.*  
*Email: [edward@essex.ac.uk](mailto:edward@essex.ac.uk)*

**Abstract:** Scheduling of vehicles in the container terminal is often studied as a static problem in the literature, where all information, including the number of jobs, their arrival time, etc., is known beforehand. The objective is generally minimizing the total travelling and/or waiting times of the vehicles. When the situation changes, for example new jobs arrive or a section of the terminal is blocked, new solutions are generated from scratch. In this paper, the problem of dynamically scheduling automated guided vehicles in the container terminals is formulated as a minimum cost flow problem. This problem is then solved by a novel algorithm, NSA+, which extended the standard Network Simplex Algorithm (NSA). Like NSA, NSA+ is a complete algorithm, which means it guarantees optimality of the solution if it finds one within the time available. Our software, DSAGV, employs NSA+ and can find the global optimal solutions for 3,000 jobs and 10 millions arcs in the graph model within 2 minutes on a 2.4 GHz Pentium PC. To complement NSA+, an incomplete algorithm Greedy Vehicle Search (GVS) is designed and implemented. To evaluate the relative strength and weakness of NSA+ and GVS, the two algorithms are compared for the dynamic automated vehicle scheduling problem. Both NSA+ and GVS are practical algorithms for dynamic automatic vehicle scheduling.

**Keywords:** Search Methods, Dynamic Scheduling, Network Simplex Algorithm, Optimization, Container Terminals.

## 1. INTRODUCTION

The Minimum Cost Flow (MCF) Problem is to flow resources from a set of supply nodes, through the arcs of a network, to a set of demand nodes at minimum total cost, without violating the lower and upper bounds on flows through the arcs (which represent the capacities of the arcs). The MCF problem has numerous applications in scheduling, transportation, logistics, and telecommunication. This paper has been motivated by a need to schedule a number of Automated Guided Vehicles (AGVs) in the container terminals. The container terminals components that are relevant to our problem include Quay Cranes (QC), container storage areas, Rubber Tyred Gantry Crane (RTGC) or yard crane, and a road network [see e.g. 5, 6, 10, 16, 17, 22]. A transportation requirement in a port is described by a set of jobs, where each job is characterized by the source location of a container, the target location and the time of its picking up or dropping-off on the quay side by the quay crane. Given a number of AGVs and their availability, the task is to schedule the AGVs to meet the transportation requirements.

The Network Simplex Algorithm (NSA) is an adaptation of the bounded variable of traditional primal simplex algorithm in Linear Programming. Generally, NSA is employed to solve MCF problems. The basis of NSA is represented as a rooted spanning tree of the underlying network, in which variables are represented by arcs. The method iterates towards an optimal solution by exchanging basic and non-basic arcs. At each iteration, an entering arc is selected by some pricing strategy, and the arc to leave the basis

is ascertained. The construction of a new basis tree is called pivoting. There are many strategies for selecting the entering arc. The performance of algorithm is depended on those strategies. This paper extends NSA with some modification to the NSA. The novel algorithm, NSA+, then is applied to dynamic automated guided vehicles scheduling.

The structure of this paper is as follows. Section 2 is a description of MCF problem and a literature review over NSA. After that features of NSA+ are presented. In section 3, an application of NSA+ for dynamic scheduling of automated guided vehicles in the container terminal is presented. Then, the problem is formulated and solved by NSA+. Also some outputs of running NSA+ to solve static problems are given in this section. Section 4 presents a Greedy Vehicle Search (GVS) and compares results of NSA+ and GVS in dynamic aspect. Section 5 is considered to summary and conclusion.

## 2. MINIMUM COST FLOW PROBLEM AND ALGORITHMS (NSA, NSA+)

### 2.1 Minimum Cost Flow Problem

A graph  $G = (N, A)$  consists of a set of nodes,  $N$ , together with a set of arcs,  $A$ . Given a directed network graph, let  $f_{ij}$  denote the amount of flow of some material on arc  $(i, j) \in A$  and  $b_i, i \in N$ , denote the amount of flow that enters the network at node  $i$ . If  $b_i > 0$ , the node is a source that supplies  $b_i$  units of flow. If  $b_i < 0$ , the node is a sink with a demand of  $|b_i|$  units of flow. Let there be a cost of  $c_{ij}$  per unit flow on arc  $(i, j) \in A$ . The *minimum cost flow* (MCF) problem is:

$$\begin{aligned} & \text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} \cdot f_{ij} \\ & \text{Subject To} \quad \begin{cases} b_i + \sum_{j:(j,i) \in A} f_{ji} = \sum_{j:(i,j) \in A} f_{ij} ; \text{ for all } i \in N \\ m_{ij} \leq f_{ij} \leq M_{ij}, \text{ for all } (i, j) \in A \end{cases} \end{aligned}$$

These constraints show that the flows must be feasible and conserve each node. For feasible flows to exist, the MCF problem must also have  $\sum_{i \in N} b_i = 0$ .

Note that the MCF problem is a special form of Linear Program. The constraints of a Linear Program are of the form  $Ax = b$ , where

$$(A)_{ik} = \begin{cases} +1 & \text{node } i \text{ is start of } k^{\text{th}} \text{ arc;} \\ -1 & \text{node } i \text{ is end of } k^{\text{th}} \text{ arc;} \\ 0 & \text{otherwise} \end{cases}$$

### 2.2 Spanning Tree Solutions and Optimality Conditions

In the network simplex algorithm, it is assumed that the network is connected. Every connected network has a spanning tree [1]. A spanning tree solution,  $T$ , is a tree that can be constructed as follows:

- Given  $n$  as the number of nodes in the graph, pick a set  $T \subset A$  of  $n-1$  arcs forming a spanning tree. The remaining arcs can be divided into the two sets  $L$  and  $U$ .
- Set  $f_{ij} = m_{ij}$  for each arc  $(i, j) \in L$  and  $f_{ij} = M_{ij}$  for each arc  $(i, j) \in U$ .
- Use the flow conservation constraints to determine the flows  $f_{ij}$  for arcs  $(i, j) \in T$ . The algorithm begins by determining the flows on arcs incident to leaves of the tree  $T$ . Subsequently it determines the flows on other arcs of  $T$ . A spanning tree solution with  $m_{ij} \leq f_{ij} \leq M_{ij}$  is a feasible spanning tree solution.

**Theorem 1 [1]:** Every basic feasible solution corresponds to a feasible spanning tree. Flow on non-basic arcs must be either  $m_{ij}$  or  $M_{ij}$ .

**Theorem 2 [1]:** A spanning tree structure  $(T, L, U)$  is optimal for the minimum cost flow problem if it is feasible and for some choice of node potential  $\pi$  the arc reduced costs satisfy the following conditions:

$$\begin{aligned}\bar{c}_{ij} &= C_{ij} - \pi(i) + \pi(j) = 0 \text{ for all } (i, j) \in T \\ \bar{c}_{ij} &= C_{ij} - \pi(i) + \pi(j) \geq 0 \text{ for all } (i, j) \in L \\ \bar{c}_{ij} &= C_{ij} - \pi(i) + \pi(j) \leq 0 \text{ for all } (i, j) \in U\end{aligned}$$

## 2.3 The Algorithm NSA

In network simplex algorithm, the linear algebra of the original simplex algorithm is replaced by simple network operations. With simple network operations, the minimum cost flow problem can be solved more than 100 times faster than equivalently sized Linear Programs. Ahuja, Magnanti, and Orlin [1] described the network simplex algorithm and gave pseudo-codes and hints. Helgason and Kennington [9] provided detailed information on an implementation. Here, a short description of the algorithm is presented.

The network simplex algorithm maintains a feasible spanning tree structure at each iteration and successfully transforms it into an improved spanning tree structure until it becomes optimal. The algorithm in figure 1 specifies steps of the method [1,12] .

To create the initial or Basic Feasible Solution (BFS), an artificial node 0 is appended to the graph. It provides one way of obtaining an initial spanning tree structure for a connected graph. It can be assumed that for every node  $j \in N - \{0\}$ , the network graph contains arc  $(0, j)$  and  $(j, 0)$ , with sufficiently large costs and capacities. The initial tree  $T$  is constructed as follows: each node  $j$ , other than node 0, is examined one at a time. If  $b(j) \geq 0$ , the algorithm includes  $(j, 0)$  in  $T$  with a flow value of  $b(j)$ . If  $b(j) \leq 0$ , it includes arc  $(0, j)$  in  $T$  with a flow value of  $-b(j)$ . The set  $L$  consists of the remaining arcs, and the set  $U$  is empty [1].

Appending the entering arc,  $(k, l)$ , to the spanning tree forms a unique cycle,  $W$ , with the arcs of the basis. In order to eliminate this cycle, one of its arcs must leave the basis. The cycle is eliminated when we have augmented flow by a sufficient amount to force the flow in one or more arcs of the cycle to their upper or lower bounds. By augmenting flow in a negative cost augmenting cycle, the objective value of the solution is improved. The first task in determining the leaving arc is the identification of all arcs of the cycle. The flow change is determined by the following equation:

$$\theta = \min \{ f_{ij} \text{ for all } (i, j) \in W \} .$$

The leaving arc is selected based on cycle  $W$ . The substitution of entering for the leaving arc and the reconstruction of new tree is called a pivot. After pivoting to change the basis, the reduced costs  $\bar{c}_{ij} = c_{ij} - (\pi_i - \pi_j)$  for each arc  $(i, j) \notin T$  is calculated. Note that  $\bar{c}_{ij} = 0$  for all arcs  $(i, j) \in T$  by construction. If  $\bar{c}_{ij} \geq 0$  for all  $(i, j) \in L$  and  $\bar{c}_{ij} \leq 0$  for all  $(i, j) \in U$  then the current basic feasible solution is optimal. Otherwise, an arc  $(i, j)$  where there is a violation should be chosen and operations of the algorithm should be repeated.



Different strategies are available for finding an entering arc for the basic solution. These strategies are called pricing rules. The performance of the algorithm is affected by these strategies. A literature review over these strategies is given below:

The standard textbook [1] provided a detailed account of the literature on those strategies. We now briefly review this literature. Goldfarb and Reid (1977) proposed a steepest edge pricing criterion. Instead of selecting an entering arc on the basis of the reduced cost, it is selected by taking into account the changes in all arcs. Another candidate list strategy has been described by Mulvey (1978). There is a major and minor loop to select the entering arc. A limited number of favourably priced entering arcs are collected by scanning the non-basic arcs in a major iteration. In the minor iteration, the most favourably priced arc in the list is chosen to enter the basis. Grigoriadis (1986) describes a very simple arc block pricing strategy based on dividing the arcs into a number of subsets of specified size. At each iteration, the entering arc is selected from a block with most negative price. Only the arcs of one block are re-priced at any iteration. Andrew [8] studied practical implementation of minimum cost flow algorithms and claimed that his/her implementations worked very well over a wide range of problems.

In recent years, several researches have been devoted on network simplex algorithm. Muramatsu [14] used a primal-dual symmetric pivoting rule and proposed a new scheme in which the algorithm can start from an arbitrary pair of primal and dual feasible spanning tree. Eppstein [7] presented a clustering technique for partitioning trees and forests into smaller sub-trees or clusters. This technique has been used to improve the time bounds for optimal pivot selection in the primal network simplex algorithm for minimum-cost flow problem. Lobel [13] developed some pricing rules to select an entering arc, a mixture of arc block and hot-list. In that pricing rule, a basket of violated arcs are collected in each iteration. A general pricing scheme for the simplex method has been proposed by Istvan [11]. He/she claimed that it creates a large flexibility in pricing and applicable to general and network simplex algorithms. Ahuja et al [2] developed a network simplex algorithm with  $O(n)$  consecutive degenerate pivot. He presented an anti-stalling pivot rule, based on concept of *strongly feasible* spanning tree. The basis structure  $(T, L, U)$  is *strongly feasible* if we can send a positive amount of flow from any node to root along arcs in the spanning tree without violating any of the flow bounds. An equivalent way of stating this property is that no upward pointing arc of the spanning tree can be at its upper bound and no downward pointing arc can be at its lower bound.

## 2.4 The Algorithm NSA+

NSA+ is an extension of NSA. Compared with the standard version of NSA, it has two features. Firstly, it deals with the concept of strongly feasible solution. Secondly, a mixture of heuristic approach and memory technique are used in NSA+. The first one is not a new, but a combination of the two features provides a higher speed. These features are briefly explained below.

The first feature is related to maintaining the strongly feasible basis at each iteration. At the beginning, NSA+ chooses a strongly feasible solution. In order to do this, the initial basic solution, which was described in the previous section, is strongly feasible. In each pivot, the algorithm also selects the leaving arc appropriately so that the next basis would be strongly feasible. Suppose that the entering arc  $(k, l)$  is at its lower bound and node  $w$  is the first common predecessor of nodes  $k$  and  $l$ . Let  $W$  be the basis cycle formed by adding arc  $(k, l)$  to the basis tree. This cycle consists of the basis path from node  $w$  to node  $k$ , the arc  $(k, l)$ , and the basis path from node  $0$  (root of the spanning tree) to node  $w$ . The algorithm sends a maximum possible amount of flow in the cycle without violating any of the lower and upper bound constraints on arcs. An arc which blocks further increase of flow in the basis cycle is called a blocking arc. If the blocking arc is unique, then it leaves the basis. If there are more than one blocking arcs, then the algorithm selects the leaving arc to be the last blocking arc encountered in traversing  $W$  along its orientation starting at node  $w$ . Due to degeneracy, cycling may occur in the network simplex algorithm.

By maintaining strongly feasible bases due to Cunningham (1976, 1979), cycling can be prevented without restrictions on the entering arc [1].

The second feature is concerned with the entering arc. The arcs in graph are divided into several blocks with the same size. At each iteration, a packet of the violated arcs are collected. The capacity of the packet is more than the block's size and the most violated arcs are kept at the top of the packet. The number of most violated arcs may be a percentage of the block's size. We set the block's size and number of most violated arcs to 200 and 25, respectively. For each problem, the number of blocks is depends on the number of arcs in the graph and the block's size. In our software, the blocks are identified by a block-number and the first one is chosen randomly or by a heuristic method (based on location of the biggest cost of arcs, for example). Scanning of the arcs for violation among different blocks is chosen circularly. At each iteration, one violated arc (at most) from each block is put in the packet. At the beginning of the entering arc procedure, reduced costs of the most violated arcs in previous stage are recalculated. If they violate the optimality conditions again, they are kept in the packet. Otherwise they could be replaced by new violated arcs. Then, some new violated arcs, based on scanning of arcs from the blocks, are put into the packet as long as it has empty place. At the end of procedure, if the packet is empty, the current solution is optimal. Otherwise the packet will be sorted decreasingly, based on absolute value of the reduced costs, and the most violated arc will be chosen as the entering arc.

### 3. Scheduling Problem of Automated Guided Vehicles

In this section, scheduling problem of Automated Guided Vehicles (AGVs) in the container terminals is introduced. The problem is to deploy several AGVs in a port to carry many containers from the quay-side to yard-side or vice versa. The main reason to choosing this problem is that the efficiency of a container terminal is directly related to use the AGVs with full efficiency [see e.g. 5, 6, 10, 19, 22]. This problem is formulated as a Minimum Cost Flow (MCF) model and then solved by primal version of NSA+.

#### 3.1 Assumptions

Following are the assumptions made in this paper:

**Assumption 1:** The layout of a port container terminal is given [22]. According to a specific layout the travel time between every combination of Pickup (P) /Drop-off (D) points is provided.

**Assumption 2:** It is assumed the vehicles move with an average speed so that there are no Collisions, Congestion, Live-locks, Deadlocks [19] and breakdown problem while they are carrying the containers.

**Assumption 3:** Every AGV can transport only one container. Also it is assumed that the start location of each AGV at the beginning of process is given.

**Assumption 4:** The yard is divided to several blocks and RTGCs or yard crane resources are always available [6], i.e., the AGVs will not suffer delays in the storage yard location or waiting for the yard cranes.

**Assumption 5:** Appointment time of every job at its source (destination) on the quay side is given.

**Assumption 6:** The quay side consists of several Quay Cranes (QCs). For each QC, there is a predetermined job sequence, consisting of loading or unloading jobs, or a combination of both. For each loading (unloading) job, there is a predetermined pickup (drop-off) point in the yard, which is the origin (destination) of the job.

**Assumption 7:** There are N containers jobs in the container terminal. The source and destination of them in the port are given.

**Assumption 8:** For the dynamic aspect of the problem, it is assumed that the number of vehicles is fixed, but the number of jobs and the distance between every two points in the port may be changed.

### 3.2 Minimum Cost Flow Model of the Problem

To make a model for the problem, the following notations are used:

- $t_i$  : Appointment time of job  $i$  at the quay side (to be unloaded or dropped-off).
- $RTA_m$ : Ready time of AGV  $m$  at start location, where may be either the quay-side or yard.
- $T_{ij}$ : Travelling time from location of job  $i$  (quay-side) to location of job  $j$  (quay-side).
- $P$ : Penalty of delay if an AGV can not reach on the quay side at the appointment time.
- $TTA_{mi}$ : The travel time of the AGV  $m$  from the start location to the location of job  $i$  on the quay side.

The calculation of travel time between the jobs  $i$  and  $j$ ,  $T_{ij}$ , is illustrated by figure 2 in different cases [6]. The  $RTA_m$  should be calculated in the similar manner.

The optimal assignment of the AGVs to container jobs can be determined by solving a minimum-cost network flow problem. Cheng et al. (2003) minimized the impact of delays and waiting times of the AGVs at the quay side [6]. In this paper, our objectives are to minimize (a) the total AGV waiting time at the quay side (b) the total AGV travelling time in the route of port (c) the total lateness times to serve the jobs.

A directed graph to represent the complete network flow is suggested. Given number of jobs,  $N$ , and number of vehicles,  $M$ , in the problem, there are  $N+M+1$  node in the graph. Every container job (unloading/loading) is represented as a node in the graph. There is a node corresponding to each vehicle, capturing the state of the vehicle at the time of deployment. We also need to insert a sink node, corresponding to the end state of the vehicles, after all container jobs have been served.

To define the arcs in the graph model, the following notations are introduced:

- The job  $j$  is consistent with job  $i$  if a single vehicle can be used to serve job  $j$  after serving job  $i$ . Therefore, these jobs are consistent if:  

$$t_i + T_{ij} \leq t_j .$$
- Similarly the vehicle  $m$  is consistent with job  $i$  if:  

$$RTA_m + TTA_{mi} \leq t_i .$$

The following three types of arcs connect the various nodes in the graph:

1) There is a directed arc from a vehicle, node  $m$ , to a container job, node  $i$ . If the vehicle  $m$  is consistent with job  $i$ , cost of the arc  $(m, i)$  corresponds to the waiting and travelling time that the vehicle  $m$  incurs at the quay crane location of job  $i$ . However, in practical situation, it may not be possible for the vehicle to arrive at the quay side at the appointment time. A large penalty should be considered for this delay to avoid those inconsistent arcs.

Hence, the Waiting and Travelling (Distance) times between vehicle  $m$  and job  $i$  as well as the Lateness times to serve job  $i$  are:

$$\left\{ \begin{array}{l} \text{if job } i \text{ can be carried by AGV } m (t_i \geq RTA_m + TTA_{mi}) \rightarrow \left\{ \begin{array}{l} WT_{mi} = t_i - (RTA_m + TTA_{mi}) \\ DT_{mi} = RTA_m + TTA_{mi} \\ LT_{mi} = 0 \end{array} \right\} \\ \text{if job } i \text{ can not be carried by AGV } m (t_i < RTA_m + TTA_{mi}) \rightarrow \left\{ \begin{array}{l} LT_{mi} = (RTA_m + TTA_{mi} - t_i) \\ WT_{mi} = DT_{mi} = 0 \end{array} \right\} \end{array} \right.$$

2) There is a directed arc from container job  $i$  to container job  $j$ . If the job  $j$  is consistent with job  $i$ , cost of the arc  $(i, j)$  is the waiting and travelling times that the vehicle incurs. However in the practical situation, it is impossible to serve the jobs within the specified time. Similar to the inconsistent arc

between vehicle node and containers node, a large penalty should be considered to avoid serving these sequences. Thus, the Waiting and Travelling (Distance) times of vehicles and the Lateness times to serve job node  $j$  after serving job node  $i$  are:

$$\left\{ \begin{array}{l} \text{if job } j \text{ can be served after job } i \ (t_j \geq t_i + TT_{ij}) \rightarrow \left\{ \begin{array}{l} WT_{ij} = t_j - (t_i + TT_{ij}) \\ DT_{ij} = TT_{ij} \\ LT_{ij} = 0 \end{array} \right\} \\ \text{if job } j \text{ can not be served after job } i \ (t_j < t_i + TT_{ij}) \rightarrow \left\{ \begin{array}{l} LT_{ij} = (t_i + TT_{ij} - t_j) \\ WT_{ij} = DT_{ij} = 0 \end{array} \right\} \end{array} \right.$$

3) There exists a directed arc from each of the vehicle nodes and the container nodes to the sink node, which denoted by  $s$ . These arcs show that a vehicle can remain idle after serving any number of jobs or without serving any job. Therefore, a cost of zero is assigned to these arcs:

$$\left\{ \begin{array}{l} WT_{ms} = DT_{ms} = LT_{ms} = 0; \text{Cost between vehicle } m \text{ and Sink is zero.} \\ WT_{is} = DT_{is} = LT_{is} = 0; \text{Cost between job } i \text{ and Sink is zero.} \end{array} \right.$$

### 3.3 The Mathematical Model

The MCF model in the previous section can be written in mathematical form as follows:

$$\begin{array}{l} \text{Min} \left[ w1(\sum_i \sum_j WT_{ij} f_{ij}) + w2(\sum_i \sum_j DT_{ij} f_{ij}) + P \sum_i \sum_j LT_{ij} f_{ij} \right] \\ \text{S.t.} \left\{ \begin{array}{l} \sum_j f_{ij} - \sum_j f_{ji} = 0; \ \forall i \in \text{Container Nodes} \\ \sum_j f_{ij} = 1; \ \forall i \in \text{AGV Nodes} \\ \sum_j f_{ji} = M; \ \text{for } i = \text{Sink Node} \\ f_{ij} = \begin{cases} 1 & \text{if AGV goes from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases} \end{array} \right. \end{array}$$

In the model  $f_{ij}$ ,  $WT_{ij}$ ,  $DT_{ij}$  and  $LT_{ij}$  are the flow traversing, waiting, travelling and lateness costs across arc  $a_{ij}$ , respectively. In the objective function,  $w1$  and  $w2$  are weights for the waiting and travelling times of the vehicles. A large value,  $P$ , is considered to avoid the inconsistent arcs. The first three constraints show the flow balance for the network, and the fourth constraint decrees that all the container nodes are visited exactly once. Solving this network model generates  $M$  paths, each of which commences from a vehicle node and terminates at the sink node. Each path determines a job sequence of every vehicle.

The MCF model of the problem can be illustrated by figure 3 for two AGVs and four container jobs. In order to prevent an AGV to go directly to the sink node, every container node should be replaced by a couple of nodes. The arc between this couple (nodes 3 and 4 for job<sub>1</sub>, for example) has unit lower and upper bound. Moreover, the transition cost between these couples is zero. These conditions guarantee that every container node should be visited once only.

The problem has a huge search space and the solution should provide the optimal paths for each AGV from every vehicle node to the sink node. Given number of jobs and number of vehicles,  $N$  and  $M$  respectively, there are  $M+2*N+1$  nodes and  $M+M*N + N*(N-1) + 2*N$  in the graph model. Suppose that for some values of arc costs, the paths given by NSA+ are 1-3-4-9-10-11 and 2-5-6-7-8-11. This states that AGV 1 is assigned to serve jobs 1 and 4, and AGV 2 is assigned to serve jobs 2 and 3, respectively.

### 3.4 Experimental Results for Static Problems

To test the model and NSA+, a hypothetical port was designed. In each run, random jobs were generated. In each generated job, the sources and destinations were chosen randomly by our software (DSAGV). The parameters in table 1 were used to define the port, the objective function, the number of vehicles and generate the jobs.

We implemented DSAGV in Borland C++. Then, the software has been run to solve several random problems. The time to solve the problems has been drawn, by figure 4, according to both number of jobs and number of arcs in the graph model. Also the power estimation for those two curves has been shown on the figures. All experiments were run on a Pentium-4 2.4 GHz PC with 1 GMB RAM. As it can be seen in the figures, NSA+ can find the global optimal solution for 3,000 jobs and 10 millions arcs in the graph model within 2 minutes.

### 3.5. Complexity of NSA+

Assume that maximum flow,  $U$ , in each of the  $m$  arcs, at maximum cost,  $C$ . So there is an upper bound on the value of the objective function of a Minimum Cost Flow problem. This upper bound is given by  $mCU$ . There are two different types of pivots in the algorithm, non-degenerate and degenerate pivots. The former is bounded by  $mC$  because the number of non-degenerative pivots in the algorithm is bounded by  $mCU$  ( $U=1$  in this application). The number of degenerate pivots is at most  $n^2C$ , in which  $n$  denotes the number of nodes. A series of degenerate pivots may occur between each pair of non-degenerate pivots, and thus a bound on the total number of iterations is  $m \cdot n^2 \cdot C^2$ . Finding the entering arc is an  $O(m)$  and sorting the packet is  $O(K \cdot \text{Log}K)$  operation ( $K$  is size of packets,  $K=225$ ). Finding the cycle, amount of flow change, leaving arc and updating the tree are  $O(n)$  operations. Hence the complexity of each pivot is  $O((m + n) K \cdot \text{Log}K)$ . Based on the complexity of number of iterations and each pivot, the total complexity of this algorithm is determined as follows:

$$O((m + n)mn^2C^2K\text{Log}K)$$

In our graph model  $m = O(n^2)$ , therefore the total complexity of the network simplex plus algorithm in this application is:

$$O(n^6)$$

To confirm that result, the complexity of the algorithm has been estimated by the outputs of the software and some estimation methods. The estimation's result shows this complexity, which can be seen in figure 4.

## 4. Dynamic Scheduling of AGVs

Traditionally, scheduling problem of vehicles in the container terminal is static nature where everything is known beforehand. In reality, the problem is dynamic. New jobs arrive from time to time. Delays or breakdown of vehicles change the travelling time and availability of vehicles. NSA+ is a complete algorithm. Although it is efficient, it can only work on problems with certain limits in size. To

complement NSA+, a Greedy Vehicle Search (GVS) is designed and implemented. GVS will be useful for problems which sizes go beyond the limit of NSA+, or in dynamic scheduling where reactive responses are called for.

#### 4.1 The Greedy Vehicle Search Algorithm

In this simple search method, every time a job needs to be served. This behaves as a Taxi Service System. For any unassigned job and the list of idles AGVs, a job is assigned to a vehicle with minimum costs, including waiting and travelling times of the vehicles as well as lateness of the jobs. A block diagram describing GVS is presented in figure 5.

Given the appointment time of container job  $j$ ,  $t_j$ , the ready time of AGV  $m$  to get the start location,  $RTA_m$ , and the travel time of the AGV from its start location to the source/destination of job  $j$  on the quay side,  $TTA_{mj}$ , the objective function is calculated as:

$$Objective \ Function(m, j) = \begin{cases} w1 \times (t_j - RTA_m - TTA_{mj}) + w2 \times (RTA_m + TTA_{mj}) & \text{if } t_j > RTA_m + TTA_{mj} \\ P \times (RTA_m + TTA_{mj} - t_j) & \text{otherwise} \end{cases}$$

Note that  $w1$  and  $w2$  are the weight of waiting time and travelling time of the vehicles and  $P$  is a penalty, as section 3. The pseudo code for GVS is shown in figure 6.

We can compare this method with the graph model in section 3. This is a special case of the minimum cost flow problem with  $M$  vehicle node,  $N$  container nodes and one sink node, but every time an idle vehicle is assigned to just one of the remaining jobs.

The constraints for this simple cost flow problem are as follows:

$$S.t \left\{ \begin{array}{l} \sum_j f_{ij} = 1; \quad \forall i \in AGV \text{ Nodes} \\ \sum_j f_{ji} = M; \quad \text{for } i = Sink \text{ Node} \\ f_{ij} = \begin{cases} 1 & \text{if AGV goes from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases} \\ f_{is} = \begin{cases} 1 & \text{if } \sum_m f_{mi} = 1 \\ 0 & \text{Otherwise} \end{cases} \quad \forall i \in Container \text{ Nodes}; s = Sink \text{ Node} \end{array} \right\}$$

#### 4.2 Software Architecture for Dynamic Aspect

The architecture of the main part of the software that implements GVS is shown in figure 7. In this section, this architecture and operation are explained briefly. A similar architecture and operation for NSA+ are considered in our software.

At the start of process, the Job Generator generates a few jobs for the cranes. These jobs will be appended to the remaining jobs, which is empty at the beginning. The remaining jobs are used by the Greedy Vehicle Search and the output of this method is an individual job for every vehicle.

The software does two tasks in dynamic fashion. The first task is related to updating the vehicle's status and assigning a job to any available vehicle whereas the second one takes influence from any idle

crane. In real time processing, the travelled and waited times of every vehicle should be updated. At the same time, if the vehicle picks up the job from the quay side, the assigned job for the vehicle will be deleted and removed from the list of remaining jobs. If the job should be delivered to the crane on the quay side, it could not be removed until meeting time between the crane and the vehicle (the appointment place is on the quay side, not the yard side). The second task refers to any change in the crane's status. The Job Generator has to generate a few new jobs, when it finds out any idle crane.

### 4.3 A Comparison between GVS and NSA+

To evaluate the relative strength and weakness of GVS and NSA+ in the dynamic scheduling problem, we used randomly generated problems. Distance between every two points in the port as well as the source and destination of jobs were chosen randomly. We did a simulation for 6 hours subject to generating 5 jobs for any idle crane. Other parameters for this simulation were the same as table 1. Figures 8 to 10 demonstrate some outputs of the software during the six hours.

As we can see from figure 8, waiting times of the vehicles for GVS is significantly greater than waiting times of the vehicle in NSA+, although travelling times of the vehicles for both algorithms are almost the same during the 6 hours. The main reason for this result is that, being a complete algorithm, NSA+ finds the optimal solution for the Minimum Cost Flow problem whereas GVS is incomplete search method.

The number of jobs carried out by the end of six-hour (21,600 seconds) for both algorithms is approximately the same (see figure 9). Generally, due to the tight schedules of the quay cranes, it is undesirable for containers to be served early or too late for the appointment. Therefore, the average lateness from the appointment times is another indicator for evaluating the algorithms. Given the number of served jobs,  $N$ , the time at which the job  $i$  is served,  $ACT_i$ , and the time of job- $i$ 's appointment,  $APT_i$ , a schedule's average lateness is calculated by the following equation:

$$\text{Average Lateness} = \frac{\sum_{i=1}^N (ACT_i - APT_i)}{N}$$

Figure 10 presents the Average Lateness indicator for both NSA+ and GVS during the six-hour simulation. The figure shows that both algorithms performed well, but GVS is superior to NSA+ in average lateness. This is hardly surprising as NSA+ was not designed to minimize the average lateness measure.

### 4.4 Complexity of GVS

There is no doubt that GVS is very fast. We got some samples to show its performance. The CPU-Time to solve the problems by GVS is shown in figure 11. As we can see when the number of jobs is 10,000, it doesn't get too much CPU-Time to solve this large problem (less than 1 second). On average, for a problem with 3,000 jobs, NSA+ takes 2 minutes to generate the new solution. Due to memory problem, NSA+ did not always generate a solution for problems with 10,000 jobs. GVS found solutions in a second. Note that these observations have been collected by running the algorithms on 2.4GHz Pentium PC for 50 vehicles.

Given the number of jobs and vehicles in the problem,  $N$  and  $M$ , respectively for this algorithm, its complexity is  $O(N \cdot M)$ . It is too easy to understand this complexity from Pseudo code of the algorithm (see by figure 6).

## 4.5 Statistical Test for the Comparison

The waiting and travelling times of the vehicles in dynamic scheduling (see section 4.3), produced by NSA+ and GVS, were analysed statistically. An observation was collected every time when there were changes in the problem and the algorithms had to solve the new problem. We tested the null hypothesis that the means produced by the two algorithms were statistically indifferent. Table 2 shows the test results along with the critical values of t-distribution. Since we cared the change was positive or negative, 'One-tail' test was chosen. The Paired t-test suggests that, with a confidence level of 95%, NSA+ is better than GVS in both travelling and waiting times of the vehicles. On the other hand, GVS is statistically better than NSA+ in average lateness.

## 5. Concluding Summary

In this paper, the automated guided vehicles scheduling problem in the container terminals was formulated as a minimum cost flow and then solved by a new version of Network Simplex Algorithm (NSA+). Our software, implemented in Borland C++, running on a 2.4 GHz Pentium PC, could find the global optimal solution for 3,000 jobs and 10 millions arcs in the graph model within 2 minutes. To complement NSA+, the Greedy Vehicle Search (GVS) was designed. The two algorithms were compared on both static and dynamic problems. Given the results so far, we claim that NSA+ is efficient and effective in both waiting and travelling times of the vehicles. GVS is very fast and useful when the problem is too big for NSA+ to solve. Being an incomplete algorithm, GVS sacrifices completeness. This establishes NSA+ plus GVS as complementary algorithms for practical dynamic automatic vehicle scheduling.

## REFERENCES

1. Ahuja R.K., Magnanti T.L., Orlin J.B., (1993), "Network Flows: Theory, Algorithms and Applications". Prentice Hall.
2. Ahuja R.K., Orlin J. B., Sharma P., Sokkalingam P.T., (2002), "A network simplex algorithm with  $O(n)$  consecutive degenerate pivots". Operations Research Letters 30.
3. Ahuja R.K., Orlin J.B., Giovanni M.S., Zuddas P., (1999), "Algorithms for the simple equal flow problem", Management Science, 45(10):1440--1455.
4. Andrew V.G., (1997) "An efficient implementation of a scaling minimum-cost flow algorithm". Journal of Algorithms, 22(1):1-29.
5. Böse J., Reinert T., Steenken D., Voß S., (2000), "Vehicle Dispatching at Seaport Container Terminals Using Evolutionary Algorithms". Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, IEEE, Piscataway, DTM-IT: 1-10.
6. Cheng Y., Sen H., Natarajan K., Teo C., Tan K., (2003), "Dispatching automated guided vehicles in a container terminal", Technical Report, National University of Singapore.
7. Eppstein D., (1999) "Clustering for faster network simplex pivots", In Proc. 5th ACM-SIAM Symposium. Discrete Algorithms, pp 160--166.
8. Goldberg A.V., Kennedy, R. (1993), "An Efficient Cost Scaling Algorithm for the Assignment Problem". Technical Report, Stanford University.
9. Helgason R., Kennington J., (1995), "Primal Simplex Algorithms for Minimum Cost Network Flows," Handbook on Operations Research and Management Science Volume 7, Editors M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, North-Holland, Amsterdam, 85-133.
10. Huang Y., Hsu W.J., (2002), "Two Equivalent Integer Programming Models for Dispatching Vehicles at a Container Terminal". CAIS, Technical Report, School of Computer Engineering, Nan yang Technological University, Singapore 639798.
11. Istvan M., (2001-3), "A General Pricing Scheme for the Simplex Method", Technical Report, Department of Computing, Imperial College, London.
12. Kelly D.J., O'Neill G.M., (1993), "The Minimum Cost Flow Problem and The Network Simplex Solution Method", Master Degree Dissertation, University College, Dublin.
13. Löbel A., (2000), "A Network Simplex Implementation", Technical Report, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB).
14. Masakazu M., (1999), "On network simplex method using primal-dual symmetric pivoting rule", Journal of Operations Research of Japan.



15. Murty K.G., Jiyin L., Yat-Wah W, Zhang C, Maria C.L. Tsang, Richard J. Linn. (2002), "DSS (Decision Support System) for operations in a container terminal". Decision Support System.
16. Patrick J.M., Dekker R., (2003), "Operations research supports container handling", Technical Report EI 2001-22, Erasmus University of Rotterdam, Econometric Institute.
17. Patrick J.M., Wagelmans P.M., (2001), "Dynamic scheduling of handling equipment at automated container terminals", Technical Report EI 2001-33, Erasmus University of Rotterdam, Econometric Institute.
18. Patrick J.M., Wagelmans P.M., (2001), "Effective algorithms for integrated scheduling of handling equipment at automated container terminals", Technical Report EI 2001-19, Erasmus University of Rotterdam, Econometric Institute.
19. Qiu L., Hsu W.-J., Huang S.-Y and Wang H. (2002), "Scheduling and Routing Algorithms for AGVs: a Survey". International Journal of Production Research, Taylor & Francis Ltd, vol. 40, no. 3, pp (745-760).
20. Sen H., (2001), "Dynamic AGV-Container Job Deployment". Technical Report, HPCES Programme, Singapore-MIT Alliance.
21. Tsang E.P.K., (1995), "Scheduling techniques -- a comparative study", British Telecom Technology Journal, Vol.13, No.1., Martlesham Heath, Ipswich, UK.
22. Wook B.J., Hwan K.K., (2000), "A pooled dispatching strategy for automated guided vehicles in port container terminals", International Journal of management science, Vol 6, No 2.

## Table and Figures

*Table 1: Value of Parameters for the simulation*

| Description of the Parameters                                         | Values     |
|-----------------------------------------------------------------------|------------|
| Number of Vehicles in the port                                        | 50         |
| Number of Quay Cranes                                                 | 7          |
| Number of Blocks in the yard (Storage area inside the port)           | 32         |
| Time Window of the Cranes                                             | 120 second |
| Weight of waiting Times for the Vehicles in the objective function    | 1          |
| Weight of travelling Times for the Vehicles in the Objective Function | 5          |
| P as the penalty (See mathematical formulation)                       | 10000      |

*Table 2: The result of T-Test for the difference between NSA+ and GVS*

| Statistical Parameters                | Total Waiting Times of the Vehicles | Total Travelling Times of the Vehicles | Average lateness from the appointment times |
|---------------------------------------|-------------------------------------|----------------------------------------|---------------------------------------------|
| Number of Observations                | 721                                 | 721                                    | 721                                         |
| t-test (Paired Two Sample For Means ) | -43.40547441                        | -43.5902651                            | 73.68094065                                 |
| Degree of Freedom                     | 720                                 | 720                                    | 720                                         |
| Critical t-value                      | -1.646972                           | -1.646972                              | -1.646972                                   |

**Algorithm Network Simplex Method**

**Begin**

*Generate Initial BFS; (T, L, U)*

$(k, l) \leftarrow \text{Entering Arc } \in \{L + U\}$

**While**  $(k, l) \neq \text{NULL}$  **Do**

*Find Cycle*  $W \in \{T + (k, l)\}$

$\theta \leftarrow \text{Flow Change}$

$(p, q) \leftarrow \text{Leaving Arc } \in W$

*Update Flow in W by  $\theta$*

*Update BFS; Tree T*

*Update node potentials*

$(k, l) \leftarrow \text{Entering Arc } \in \{L + U\}$

**End while**

**End Algorithm**

Figure 1: The Network Simplex Algorithm (NSA)

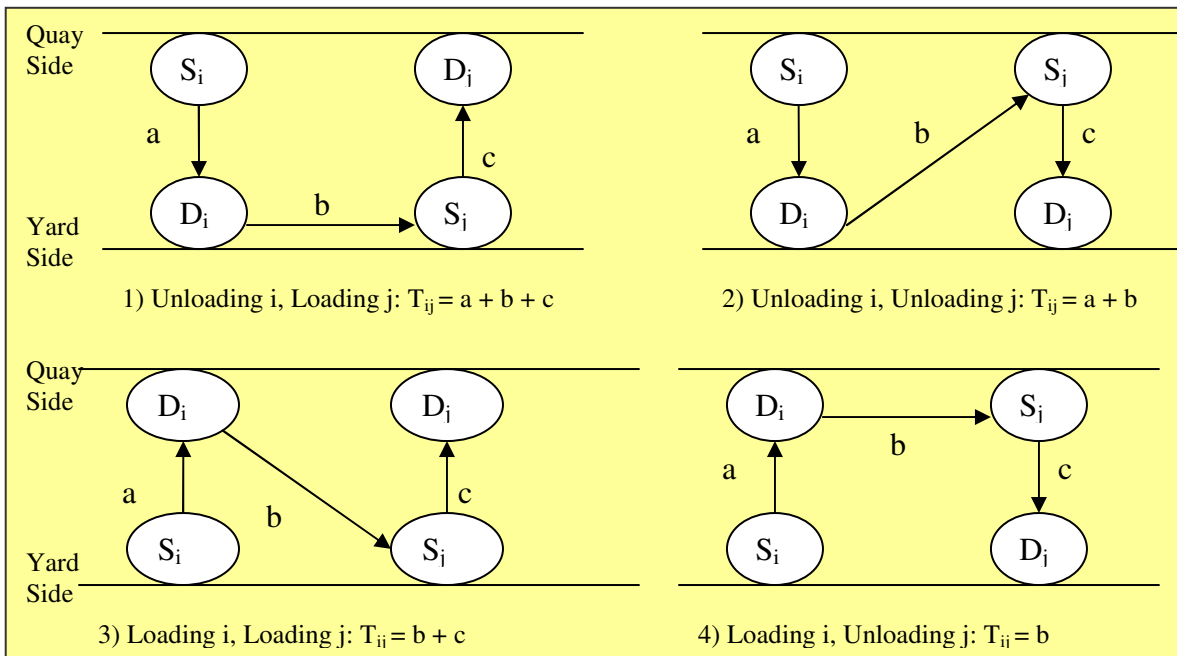


Figure 2: Travelling time computations between two jobs.

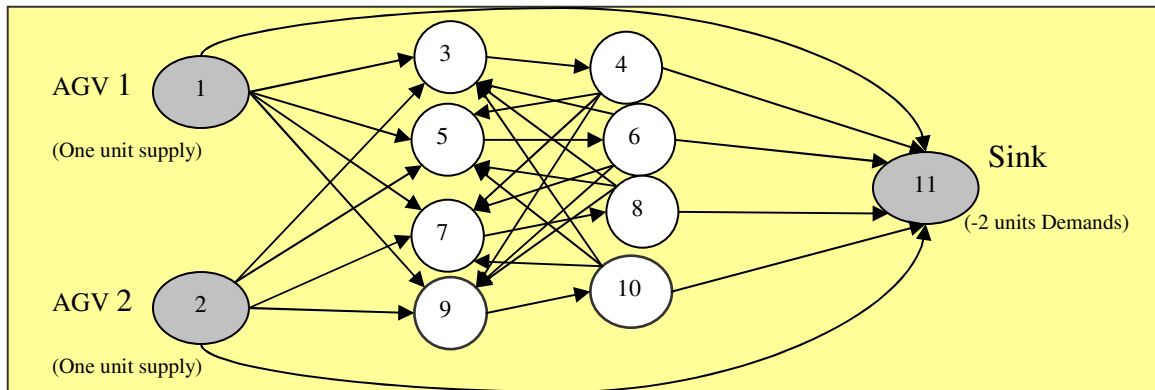


Figure 3: An example of network flow model for AGV scheduling.

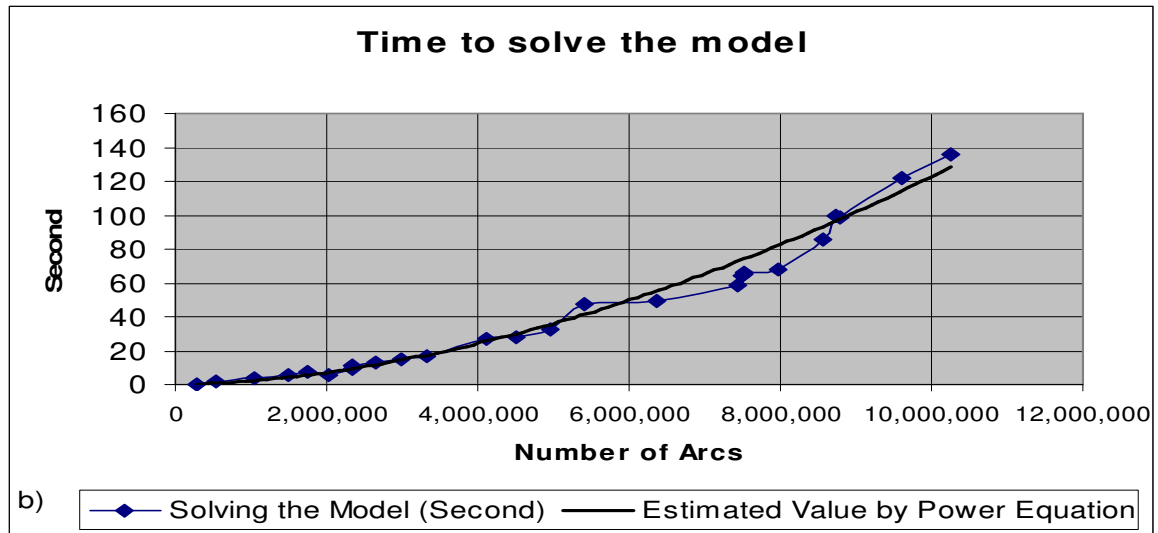
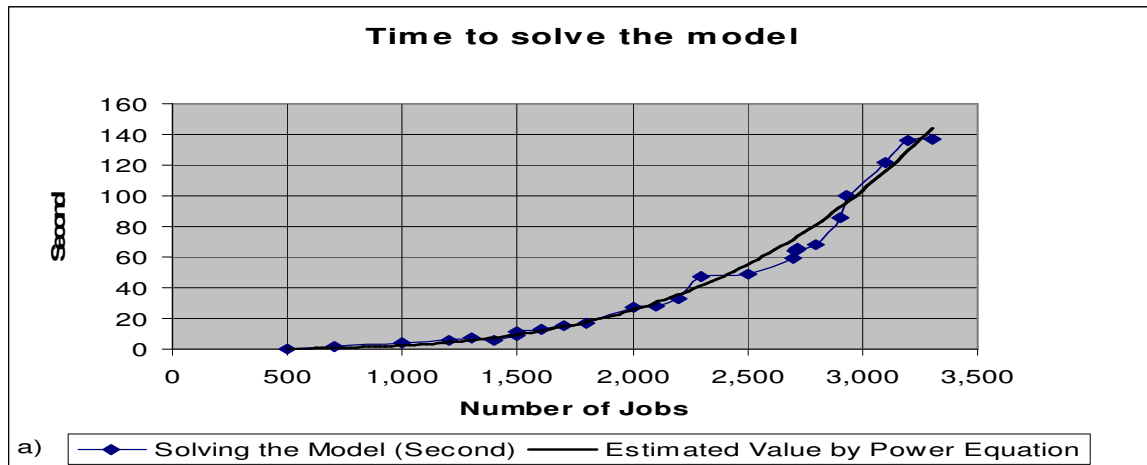


Figure 4: CPU-Time to solve the graph model by NSA+, a) based on number of Jobs, b) based on number of Arcs

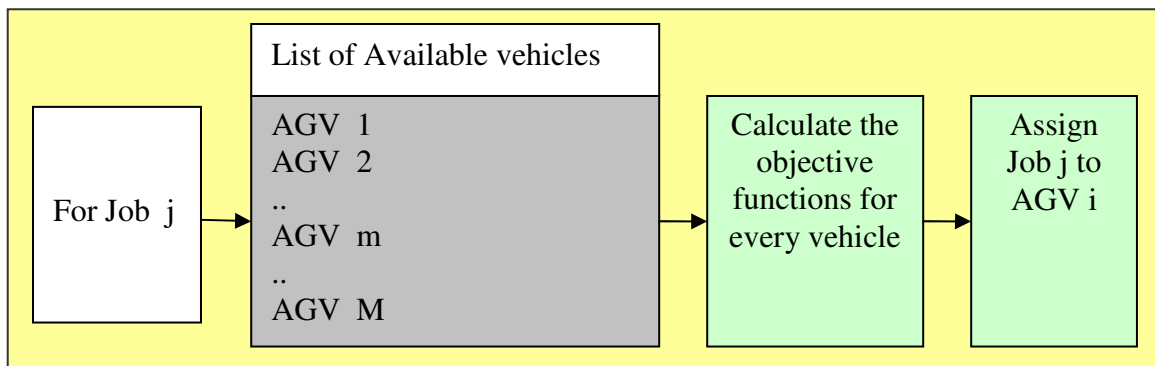


Figure 5: Block Diagram of Greedy Vehicle Search in static aspect.

```

If there is any remaining job and there is any idle vehicle
  Calculate the objective function (v, j): the travelling times and waiting times.
  For v=1, 2, #Idle vehicles; for j=1, 2,..., #Remaining jobs
Else
  Stop
End If
Again: Select a vehicle m and a job j with the minimum cost.
  Assign the vehicle m to the job j.
  Remove the vehicle m from the list of idle vehicles and the job j from the remaining job.
  If there is any idle vehicle and any left jobs
    Go to Again
  End If
Stop

```

Figure 6: Pseudo code of Simple Heuristic Search

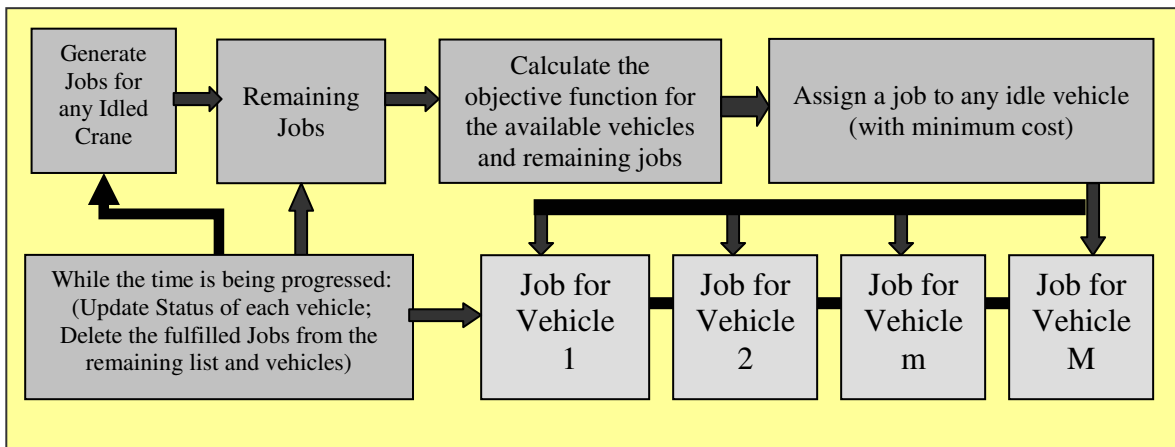


Figure 7: Block Diagram of the Greedy Vehicle Search in dynamic aspect

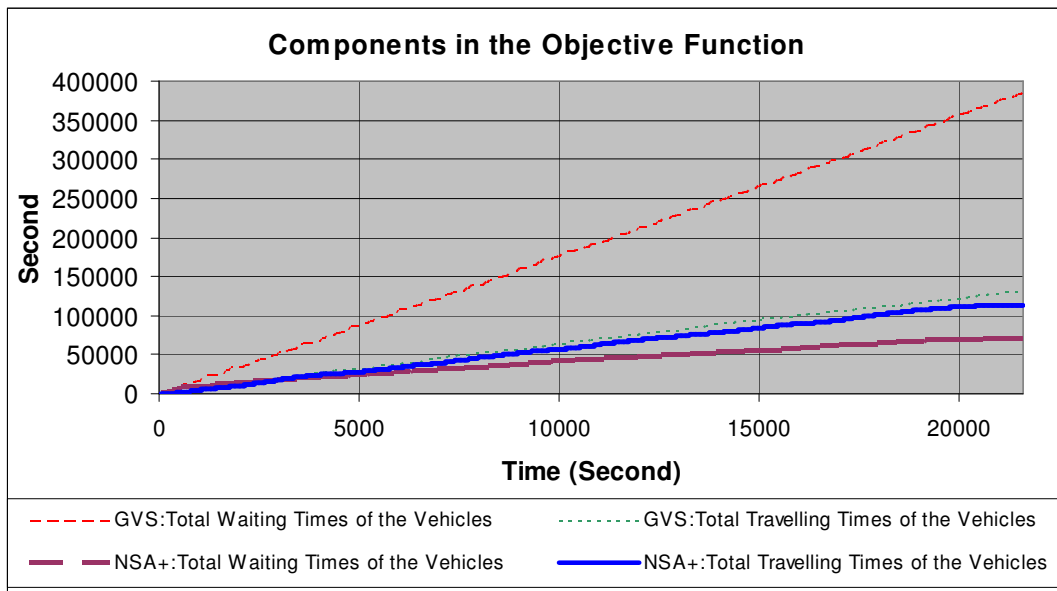


Figure 8: A comparison of Travelling and Waiting Times of the Vehicles

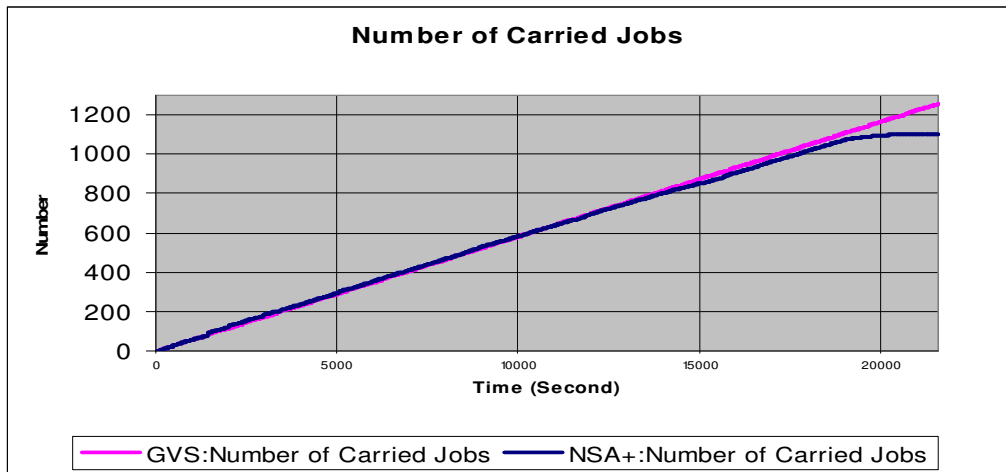


Figure 9: The number of carried jobs by two algorithms during 6 hour simulation

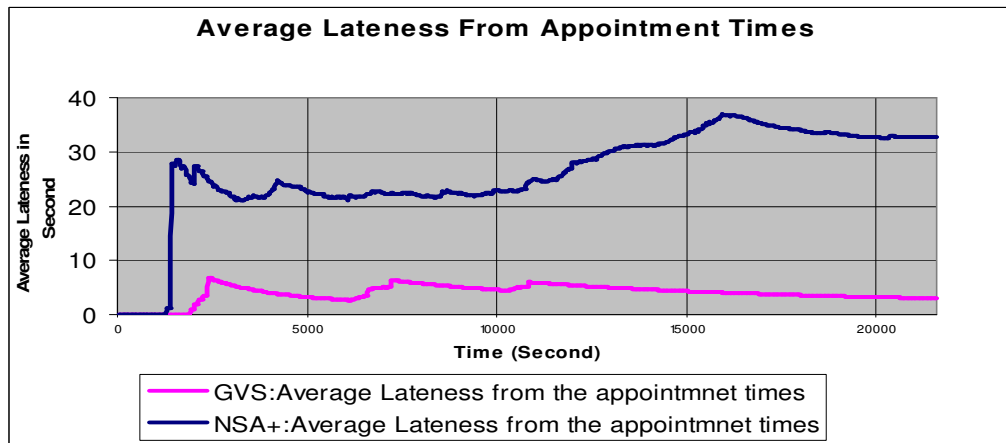


Figure 10: Average Lateness from the appointment time

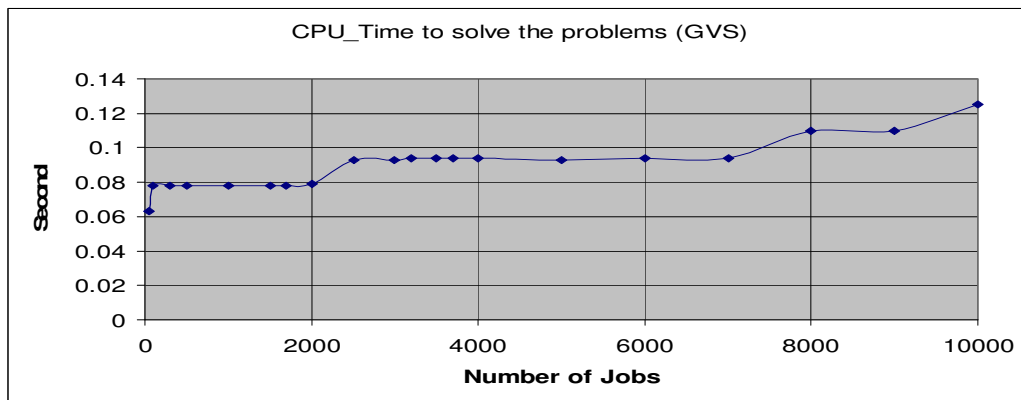


Figure 11: CPU-Time to Solve the Problem by GVS

# **Transport Scheduling**

*Abstracts*

# On a certain berth scheduling problem

Jacek Błażewicz, Maciej Machowiak<sup>1</sup>    T.C. Edwin Cheng<sup>2</sup>    Ceyda Oğuz<sup>3</sup>

---

## 1 Introduction

Berth allocation problem is to determine the berthing time and the position of each ship along the quayside. This is among the most important decision problems in a port container terminal since a good allocation of berths to the incoming ships will enhance ship owners' satisfaction and increase terminal productivity, leading to higher revenues.

We model the *berth allocation* problem (see, for example, [6, 7, 10, 11]) as a moldable task scheduling problem by considering the relation between the number of quay cranes and the berthing time. Moldable tasks form one type of parallel tasks that can be processed simultaneously on a number of parallel processors for which the processing times are a function of the number of processors assigned. The number of processors assigned to the moldable tasks is determined at the time of scheduling and will not be changed until the tasks complete their processing. In our model the quay cranes arranged along the berths are the processors and ships are the tasks to be processed by consecutive processors without preemptions. The aim in this model is to minimize the idle time on processors so as to increase the utilization of the berths. The necessity of minimizing *turn-around time* of the ships for both the port operators and the ship owners establishes the validity of this objective. From the moldable task scheduling point of view, this will be achieved by minimizing the maximum completion time of all tasks, that is the schedule length. The moldable task scheduling model was first proposed by [12] and later studied by [8] and [9].

## 2 Problem formulation

Our approach is to consider the berth allocation problem as a non-preemptable moldable task scheduling problem. This approach can simultaneously increase the utilization of quay cranes, shorten the turn-around time of ships, and decrease the waiting time of the containers.

The moldable task scheduling problem can be stated formally as follows: we consider a set of  $m$  identical *processors* (quay cranes) used for executing the set  $\mathcal{T}$  of  $n$  independent, non-preemptable (i.e. each task is processed by a constant number of processors from its start to completion) *moldable tasks* (ships). Each moldable task (MT) has to be executed for the period

---

<sup>1</sup>maciej.machowiak@cs.put.poznan.pl. Institute of Computing Science, Poznan University of Technology, Poland

<sup>2</sup>lgtcheng@polyu.edu.hk. Department of Logistics, The Hong Kong Polytechnic University, Hong Kong SAR

<sup>3</sup>coguz@ku.edu.tr. Department of Industrial Engineering, Koç University, Turkey

of  $t_i(r)$  (*processing time*) on  $r$  processors, where  $1 \leq r \leq m$ . The number of processors allotted to a task is unknown in advance. The above property distinguishes moldable tasks from the multiprocessor tasks (*size model*), considered for example in [1], where the number of processors allotted to each task is known. Also this property differentiates moldable tasks from malleable tasks (see, for example, [2]) for which the number of processors allotted to a task is also unknown but may change during their execution. As a result, the *processing speed* of a moldable task depends on the number of processors allotted to it. We let  $f_i(r)$  denote the processing speed function for task  $i$  if  $r$  processors are used to execute task  $i$ . The dependence of a moldable task processing time on a number of processors allotted is given as a discrete function; that is, it takes values at the integer points only. The criterion assumed is the schedule length, which is denoted by  $C_{max}$  with  $C_{max} = \max_i \{C_i\}$ , where  $C_i$  denotes the completion time of task  $i$ .

The problem of scheduling independent MT without preemption is NP-hard ([4]), thus, suboptimal algorithms were looked for. A survey of moldable tasks with discrete processing time functions is given by [3] and general moldable task scheduling problems with precedence constraints are reviewed by [5].

From a different point of view, we may consider a problem in which the processors represent a continuously divisible renewable resource bounded from above ([13]). In this problem the processor (resource) allocation is not required to be integer. In [2] it was shown that an optimal solution to such a problem in which processing speed functions are interpolated by piecewise linear functions between integer points (P-CNTN problem) can be found in polynomial time, assuming that the processing speed functions are all concave. In what follows, we will refer the optimal schedule length for such a solution as  $C_{cont}^*$  and will use it to construct a suboptimal solution for our discrete berth scheduling problem in question. (The procedure for solving the continuous version of the problem will not be discussed here. We refer the readers to the relevant papers by [13] and [2]).

### 3 Suboptimal algorithm and computational experiments

We propose a suboptimal algorithm to solve the moldable task scheduling problem, which was used above to model the berth allocation problem. The proposed algorithm constructs schedules for concave discrete processing speed functions with a very good average behavior. It starts from the continuous version of the problem (P-CNTN, i.e. with a piecewise linear interpolation of the discrete function) and transforms the schedule obtained from the continuous version into a feasible schedule for the discrete MT model. In the proposed algorithm, a rounding scheme for a non-integer allocation is firstly described. Next, the tasks are thoroughly packed using several steps of rounding off. These steps allow for a good average behavior, as is demonstrated in the computational experiments, for a wide range of task and processor parameters.

To evaluate the average behavior of the Algorithm we used the following measure:

$$Relative\ error = C'_{max} / C_{cont}^*$$

where  $C'_{max}$  is the schedule length obtained by the Algorithm and  $C_{cont}^*$  is the optimal schedule length of the continuous solution.

Clearly, the value  $C_{cont}^*$  is the lower bound on the optimal schedule length  $C_{max}^*$  for moldable tasks, thus, *Relative error* indicates properly the behavior of the Algorithm.



Task processing times  $t_i(1)$  have been generated from a uniform distribution in the interval  $[1..100]$ . Processing speed functions have been chosen as  $f_i(r) = r^a, 0 \leq a \leq 1$ . For  $n$  equal 16, we varied the value of  $m$  between 2 and 48 to see the effect of changing a number of processors on the average performance of the Algorithm under different processing speed functions. Then for  $m$  equal to 8, we varied  $n$  between 2 and 32 to see the influence of changing a number of tasks on the average performance of the Algorithm under different processing speed functions, (see Figure 1).

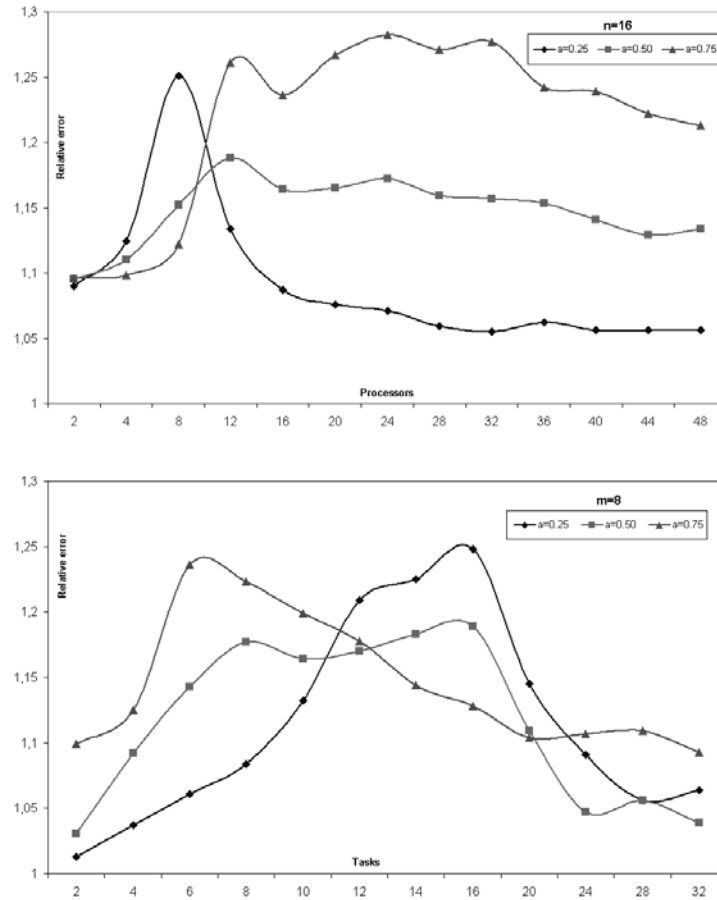


Figure 1: An influence of different shapes of processing speed functions on relative error when a number of tasks is constant ( $n=16$ , upper figure) or a number of processors is constant ( $m=8$ , lower figure).

## References

- [1] Błażewicz, J., Drabowski, M., Węglarz, J. (1986): Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers*, 35, 389–393.

- [2] Błażewicz, J., Kovalyov, M.Y., Machowiak, M., Trystram, D., Węglarz, J. (2004): Malleable task scheduling to minimize the makespan. *Annals of Operations Research*, 129, 65–80.
- [3] Drozdowski, M. (2004): Scheduling parallel tasks — algorithms and complexity. In J. Y-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, 25-1–25-25.
- [4] Du, J., Leung, J.Y-T. (1989): Complexity of scheduling parallel tasks systems. *SIAM Journal on Discrete Mathematics*, 2, 473–487.
- [5] Dutot, P-F., Mounié, G., Trystram, D. (2004): Scheduling parallel tasks — algorithms and complexity. In J. Y-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, 26-1–26-24.
- [6] Imai, A., Nishimura, E., Papadimitriou, S. (2003): Berth allocation with service priority, *Transportation Research Part B*, 37, 437–457.
- [7] Lim, A. (1998): The berth planning problem, *Operations Research Letters*, 22, 105–110.
- [8] Ludwig, W. T. (1995): *Algorithms for scheduling malleable and nonmalleable parallel tasks*. PhD thesis, University of Wisconsin - Madison, Department of Computer Sciences.
- [9] Mounié, G., Rapine, C., Trystram, D. (1999): Efficient approximation algorithms for scheduling malleable tasks. In *Eleventh ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, ACM, 23–32.
- [10] Park, Y-M., Kim, K.H. (2003): A scheduling method for berth and quay cranes. *OR Spectrum*, 25, 1–23.
- [11] Peterkofsky, R.I., Daganzo, C.F. (1990): A branch-and-bound solution method for the crane scheduling problem, *Transportation Research Part B*, 24, 159–172.
- [12] Turek, J., Wolf, J., Yu, P. (1992): Approximate algorithms for scheduling parallelizable tasks. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 323–332.
- [13] Węglarz, J. (1982): Modelling and control of dynamic resource allocation project scheduling systems. In S. G. Tzafestas (ed.), *Optimization and Control of Dynamic Operational Research Models*, North-Holland, Amsterdam.

# FORECASTING FLIGHT SCHEDULE ROBUSTNESS

## *Extended Abstract*

Edmund Burke<sup>1</sup>, Patrick De Causmaecker<sup>2</sup>, Geert De Maere<sup>1</sup>, Jeroen Mulder<sup>3</sup>, Marc Paelinck<sup>3</sup>

<sup>1</sup>University of Nottingham, UK, <sup>2</sup>Kaho Sint Lieven, Belgium, <sup>3</sup>KLM Airlines, The Netherlands

Key words: airline schedule robustness, machine learning, feature selection, hybrid model

The revenue and performance of a flight schedule is highly dependent on its robustness. Robustness can be thought of as the flexibility available within a schedule to enable a user to recover from disruptions to that schedule. A more detailed description of robustness can be found in [8]. A lack of robustness in a schedule might result in one delay leading to a series of *knock on* delays which can seriously affect the smooth operation of an airline. In order to mitigate against this, airline companies want to have accurate estimates of a schedule's robustness before that schedule is brought into operation. Regular evaluation during the scheduling process enables schedule operators to improve the schedule quality by adding *slack* where necessary and by incorporating strategic patterns – for instance a favorable slack distribution - into the schedule. A more detailed description of robust scheduling within the airline industry can be found in [1, 2].

Our full paper will present custom-built and generic models which have been developed to evaluate schedule robustness. This work builds upon some of the conclusions arising from research carried out by Lee et al. [7], Schumacher [10], Rosenberger et al. [9] and Farrington et al. [5] on stochastic modeling which provides schedule operators with a detailed what-if analysis of the schedule. However, although stochastic models provide good estimates of schedule robustness, operators often find them unsatisfactory because of their heavy computational load. This prevents them from performing schedule evaluations on a regular basis. Our method represents a much quicker approach to help operators to build better models. Moreover, our initial experiments suggest that the approach is also more accurate but this hypothesis will be rigorously evaluated and discussed in the full paper.

We explore a prediction of robustness which is based upon certain robustness *features* which are drawn from schedule operators at KLM and from recent papers in the literature [1, 2]. This feature-based prediction enables us to forecast schedule robustness. The robustness features in the flight schedule which we are exploring include:

- slack distribution
- likeliness of delay propagation
- number of redundant paths [1]
- number of swap options
- presence of strategic patterns
- number of meeting points in the schedule

The goal is that these features (and others) will be identified and translated into mathematical parameters called *explanatory variables*. The selection of a relevant subset is carried out using wrapper feature selection [6] on historic real world data from KLM. In this iterative process an induction algorithm [6] is used to train a classifier using a selected subset of features as input. Model evaluation is carried out using n-fold cross-validation [6] on the training set. The iterative process finishes once the search

algorithm ends or certain stop-criteria are met. Finally, the individual classifiers are combined into a hybrid model which leads to a better classification accuracy due to the uncorrelated errors of the individual classifiers [4]. The resulting hybrid method is used to forecast performance of a particular schedule in the future based upon trends which are discernible from the data taken from recently implemented schedules and which include passenger loads, weather conditions etc.

In our analysis of KLM data, we defined parameters based on hub banks or waves in their schedule [2]. KLM typically serves a large number of transfer passengers. To deal with this, the airline creates four hub banks or waves per day into their schedules in which a significant number of aircraft arrive and take off in a short period of time. While the commercial benefit is represented in an increased number of possible connections for their passengers, banks could also lead to an increased robustness, resulting in a better performance of the schedule. A favorable distribution in time of the aircraft standing idle at KLM's hub (at Schiphol airport) during the banks could result in an increased number of swap options leading to a greater flexibility within the schedule to enable us to recover from disruptions.

The model was built using 12 operational plans representing flight schedules for KLM's European fleet. The historically measured *five minutes departure punctuality* – that is, the number of aircraft which have actually left five minutes after their scheduled departure time - is used as the evaluation criterion for schedule performance. *Bagging* (which is a method to construct a hybrid model [3, 4]) is used to hybridise our model of the individual classifiers. Two operational plans, S03-1 and S03-2, covering the summer of 2003 were used as a test set to measure the accuracy of the resulting model.

Figure 1 shows the historic and the predicted five minute departure punctuality, averaged for each weekday in the operational plan. Figure 2 represents historically measured values and the model's output averaged for each operational plan. It can be seen that the historic and predicted curve show a strong correlation, although there is a serious discrepancy with consistently lower values for the predicted curve during the first part of summer 2003. However, further analysis leads us to the conclusion that exceptional external events – such as the war in Iraq and SARS - resulted in less passenger traffic at Schiphol leading to less passenger related delays and a higher historic punctuality. It is, of course impossible to predict such events accurately. Comparing the results of our model with KLM's simulation model shows similar results. Further refinement of the model, taking into account external influences, might overcome this deficiency in precision.

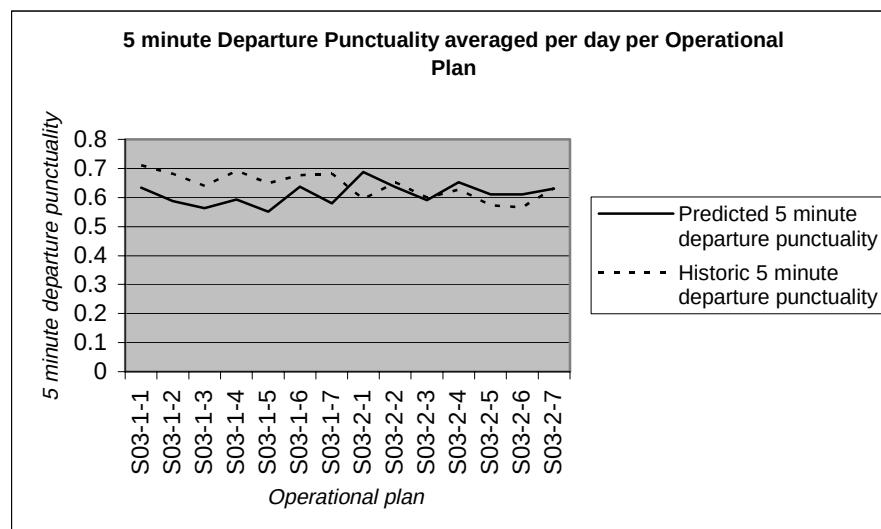


Figure 1: Historic vs. predicted 5 minute departure punctuality averaged per day, per operational plan

To conclude, we will, in the full paper, describe our forecasting methodology which is showing very promising early results. This promise will be explored further and evaluated in the full paper. The full paper will also highlight some guidelines which can be used to take schedule robustness into account when generating airline schedules.

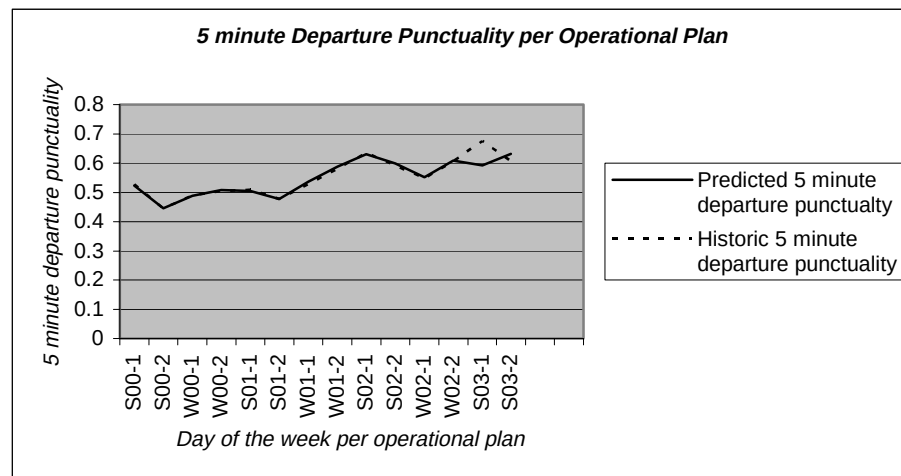


Figure 2: Historic and predicted 5 minute departure punctuality averaged per operational plan

We will also outline some future research directions which include developing more *explanatory variables* in the initial subset. It is likely that the inclusion of additional - perhaps more relevant - parameters might result in a better and more accurate model.

## REFERENCES

- [1] Ageeva Y. (2000), Approaches to Incorporating Robustness into Airline Scheduling, Master thesis, MIT
- [2] Bian F., Burke E. K., Jain S., Kendall G., Koole G.M., Landa Silva J.D., Mulder J. Paelinck M.C.E., Reeves, C. Rusdi I., Suleman M.O. (2003), Making airline schedules more robust. Proceedings of the first multidisciplinary international conference on scheduling: theory and applications, 2, 678 - 693
- [3] Breiman L. (1996), Bagging Predictors, Machine Learning, 24, No. 4, 123–140
- [4] Dietterich T. G. (1997), Machine learning research: four current directions, AI magazine, 18, No. 4, 97- 136
- [5] Farrington P. A., Nembhard H. B., Sturrock D. T., Evans G. W. (1999), Proactive flight schedule evaluation at Delta Airlines, Proceedings of the 1999 Winter Simulation Conference
- [6] John G. H., Kohavi R. (1997), Wrappers for feature subset selection, Artificial Intelligence 97, No. 1 - 2, 273-324
- [7] Lee L. H., Huang H. C., Lee C., Chew E. P., Jaruphongsa W., YongY. Y., Liang Z., Leong C. H, Tan Y. P., Namburi K., Johnson E., Banks J. (2003), Discrete event simulation model for airline operations: SIMAIR, Proceedings of the 2003 winter simulation conference
- [8] Leus R. (2003), The generation of stable project plans, Complexity and exact algorithms, PhD thesis at the university of Leuven, Faculty of applied economics.
- [9] Rosenberger. J. M., Schaefer A. J., Goldsman D., Ellis L., A. Johnson, Kleywegt J., Nemhauser G. L. (2002), A Stochastic Model of Airline Operations, Transportation Science, 36, No. 4, 357 - 378
- [10] Schumacher B. (1999), Proactive flight schedule evaluation at delta air lines, Proceedings of the 1999 Winter Simulation Conference

# Scheduling the Production and Transportation Operations subject to Dynamic Arrivals

*C. Fan, M. Kumar, L. Lei and J. Yang<sup>+</sup>*  
*Rutgers Business School, Rutgers University*  
*<sup>+</sup>New Jersey Institute of Technology*

We study the problem of coordinating the operations of a capacitated production and transportation process subject to dynamic changes in customer demand and a fixed production setup schedule. The objective is to minimize the deviation from a target supply plan and the cost of truck shipping and inventory over a T-period ( $T > 1$ ) planning horizon, where the shipping cost is charged against the number of truck-trips performed between the plant and many customer locations over a wide geographical region.

The results of this study adds the transportation and distribution operations into scheduling with dynamic arrivals for which most existing results are focused on machine scheduling. We prove that, for  $T=2$ , the problem can be solved efficiently via three exclusive cases, where two of them have closed-form solutions and one can be reduced to and solved as a binary knapsack problem. We analyze the complexity of the general case and propose a heuristic dynamic scheduling algorithm that finds a local optimal shipping schedule with a minimal deviation from the target supply plan. The worst-case error bound is analyzed for a set of special cases, and empirical observations on the gaps between the heuristic solution and a lower bound on the optimal solution are reported.

# BUILDING ROBUSTNESS INTO TRAIN DRIVER SCHEDULES

Ann S. K. Kwan, Ignacio Laplagne and Raymond S. K. Kwan

*School of Computing, University of Leeds, Leeds, LS2 9JT, UK*

{ann, ignacio, rsk}@comp.leeds.ac.uk

**Keywords:** driver scheduling, transport, set covering problem, robustness

Driver scheduling in public transport is the problem of constructing an efficient schedule of driver shifts, each of which contains a sequence of work on one or more trains separated by breaks (Kwan, 2004). The problem is usually modelled as a (ILP) set covering problem. Train delays are becoming prominent issues in the UK; accordingly, a driver schedule might not be accepted even though it is legal and efficient, if it runs a high risk of delay. Therefore, automatic driver scheduling methods should also incorporate schedule robustness, in terms of the capability of a schedule to prevent or absorb delays, as a major objective. Related work can be found in (Ehr Gott and Ryan, 2002; Kroon and Fischetti, 2000).

Interaction with schedulers allows us to identify some aspects of a schedule that are key to enhance its capability of absorbing delays that may occur during operation. We have defined two such indicators. *Slack* is the unproductive time during signing on, mealbreak or joinup. Slack occurs naturally in a schedule; it might be added to a shift so that it conforms to some labour union rules. Moreover, slack in a shift could be used to absorb some delay when it occurs. The scheduler has to strike a balance between producing a costwise-efficient schedule and adding enough slack so that the schedule meets robustness criteria.

When a train is so delayed that the slack in the driver's shift is unable to absorb, a pragmatic way which a train crew manager would employ is to switch an available driver, say  $d$ , at the same location to take over the next train that the delayed driver was due to take on. This way the train driver can be relieved and have his/her legal break, and then take on the train that was originally assigned to  $d$ . We refer to this as a *swap opportunity*. In this paper, we build facilities into a well-established commercial crew scheduling package, TRACS (Fores et al., 2002; Wren et al., 2003), that allow the user to specify places and

time ranges where slack and/or swap opportunities should be encouraged in a schedule.

These measures of robustness are integrated into the scheduling system in two different ways. In the first, settings within the ILP solver in TRACS are changed so that the system generates a set  $S$  of schedules with minimum cost (rather than a single solution, as in the usual case). Then, solutions in  $S$  are compared in terms of the new robustness measures to select the best schedule. This approach has an advantage of letting the user –if he so desires– manually compare the solutions in  $S$  and select one; this may be especially useful if the scheduler is not able to provide a precise a priori weighting of the different indicators of cost and robustness.

The second approach involves the execution of a local search phase (LS). The initial solution is obtained by the ILP solver with the cost-based objective function; the local search is then guided by any or a combination of the robustness measures instead. The LS is conducted on an extension of the ILP formulation, which incorporates a detailed model of the so-called *windows of relief opportunities*. At the same time, some restrictions of the ILP model can be dropped during the LS phase (Laplagne et al., 2004). These two features result in an enhanced capability for the distribution of slack. Additionally, both schemes allow the scheduler to specify how much cost he is willing to sacrifice in order to add robustness into the schedule.

Implementations are tested and compared on real-life scenarios from the train operator Scotrail. The first approach has also been tested by actual train crew schedulers, with positive feedback. Results suggest that robustness can be added to driver schedules; moreover, this can sometimes be achieved without increasing their costs.

## References

- Ehrgott, Matthias and Ryan, David M. (2002). Constructing robust crew schedules with bicriteria optimization. *Journal of Multi-Criteria Decision Analysis*, 11(3):139–150.
- Fores, Sarah, Proll, Les, and Wren, Anthony (2002). TRACS II: a hybrid IP/heuristic driver scheduling system for public transport. *Journal of the Operational Research Society*, 53:1093–1100.
- Kroon, Leo G. and Fischetti, Matteo (2000). Scheduling train drivers and guards: The dutch “noord-oost” case. In *33rd Hawaii International Conference on System Sciences*.
- Kwan, Raymond S. K. (2004). Bus and train driver scheduling. In Leung, J. Y-T, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 51. CRC Press.
- Laplagne, Ignacio, Kwan, Raymond S. K., and Kwan, Ann S. K. (2004). A hybridised exact and local search method for robust train driver schedules planning. In *Proceedings of the 5th PATAT International Conference, August 2004*.
- Wren, Anthony, Fores, Sarah, Kwan, Ann, Kwan, Raymond, Parker, Margaret, and Proll, Les (2003). A flexible system for scheduling drivers. *Journal of Scheduling*, 6(5):437–455.



# **Vehicle Routing**

Full Papers

# IMPROVED PRUNING IN COLUMN GENERATION OF A VEHICLE ROUTING PROBLEM

Stephan Westphal  
and Sven O. Krumke  
*Department of Mathematics*  
*University of Kaiserslautern*  
*Germany*  
{westphal,krumke}@mathematik.uni-kl.de

**Abstract** Column generation techniques have become a widely used technique to successfully solve large (integer) linear programs. One of the keys to obtaining a practically efficient algorithm is to have a fast method to limit the pricing of new columns to a small set.

We study a large scale real-world vehicle dispatching problem with soft time windows which can be modelled as an integer linear program of set partitioning type. We develop a new pruning scheme based on matchings in order to speed up the branch-and-bound enumeration in the column generation process. Computational results on real-world data illustrate the effectiveness of the new pruning scheme.

**Keywords:** vehicle dispatching, integer linear programming, dynamic column generation, shadow price model

## 1. Introduction

The German automobile club ADAC (*Allgemeiner Deutscher Automobil-Club*) maintains a heterogeneous fleet of service vehicles in order to assist people whose cars break down on their way. Service requests arrive online and are handled in five help centers spread over Germany. The goal is to provide low operational costs and a good quality of service (Krumke et al., 2002; Hiller et al., 2004).

The general dispatching problem at ADAC is *online* (i.e., decisions have to be made on the basis of incomplete data). In this paper we are concerned with the solution of the following *offline-subproblem* (snapshot-optimization): compute an “optimal” dispatch for all currently known requests subject to all operational constraints.

Each snapshot problem can be modelled as a vehicle dispatching problem (called VDP in the sequel): an instance of VDP specifies a set of units  $U$ , and a set of events  $E$ . Each unit  $u$  has a current position  $o_u$ , a home position  $h_u$  where  $u$  is heading for at the end of its shift, a logon time  $t_u^{start}$  which marks the beginning of its shift and a shift end time  $t_u^{end}$  which can be exceeded by at most  $t_u^{max-ot}$  minutes overtime. Moreover, every unit  $u$  has a set of capabilities  $F_u$  expressing the abilities of serving the different kinds of damages. Serving a request  $e$  takes  $\delta_e$  units of time and can only be accomplished by a unit whose capabilities satisfy  $F_e \subseteq F_u$ , where  $F_e$  is the set of actions to be taken at  $e$ . The driving time of unit  $u$  from point  $a$  to point  $b$  is denoted by  $\delta_u^{ab}$ .

A feasible solution of VDP is an assignment of events to units with appropriate capabilities, as well as a tour for each unit such that all events are assigned. It is also required that the service of an event does not start before its release time, and each tour for a unit starts at its current position and ends at its home position. We will briefly refer to a feasible solution as a *dispatch*.

The overall goal is to find a dispatch which minimizes the sum of operational costs (travel, service and overtime costs for the units) and the waiting costs. Operational costs arise by the assignment of units to events. Using a vehicle costs  $c_u^{drv}$  per time unit for driving,  $c_u^{svc}$  per time unit for service and  $c_u^{ot}$  per time unit of overtime. Let  $l(t) := \max\{(t - \theta_e^d), 0\}$  be the lateness, if request  $e$  (with deadline  $\theta_e^d$ ) is served at time  $t$ . Then the corresponding waiting cost is given by a quadratic penalty function

$$\text{LateCost}(e, t) := c_1 \cdot l(e, t) \cdot (1 + l(e, t)/c_2),$$

where  $c_1$  and  $c_2$  are some constants (currently at ADAC  $c_1 = 1/10$  and  $c_2 = 2.24$ ). This particular function has been chosen for strategic reasons by the ADAC: the wish is to make it more expensive to let a customer wait who has already been waiting for a longer time than another customer whose car just broke down (Krumke et al., 2002; Hiller et al., 2004).

## 2. Model and Optimization Algorithm

In the following we present the tour based integer programming model from (Krumke et al., 2002) for the VDP. In this model for each possible tour  $R$  there is a decision variable  $x_R$ . Models of set-partitioning type are by now well-established in the vehicle routing literature (Desrochers and Desrosiers, 1992; Desrosiers et al., 1995).

Let  $\mathcal{R}$  be the set of all feasible *tours*. This set splits into the sets  $\mathcal{R}_u$  of feasible tours for each unit  $u \in U$ . A tour in  $\mathcal{R}_u$  can be described by any ordered sequence  $(u, e_1, e_2, \dots, e_k)$  of  $k$  distinct events visited by  $u$  in that order. We will use the sequence  $(u)$  to denote the “empty” tour where unit  $u$  moves directly to its home position without serving any jobs. Feasibility means that the capabilities of the unit are sufficient for all  $e_i$ , i.e.,  $F_{e_i} \subseteq F_u$  for all  $i = 1, \dots, k$ .

For all  $R \in \mathcal{R}$  we define a binary variable  $x_R$  with the following meaning:  $x_R = 1$  if and only if tour  $R$  is chosen to be in the dispatch, otherwise  $x_R = 0$ . The cost of the tour  $R$  is denoted by  $c_R$ . By  $t_R^e$  we denote the arrival time at event  $e$  in tour  $R$ . Let the arrival time of  $u$  at its home position be  $t_R^{h_u}$ . Then the cost  $c_R$  of tour  $R = (u, e_1, \dots, e_k)$  can be computed as

$$\begin{aligned} c_R &= c_u^{drv} \delta_u^{o_u e_1} + \sum_{i=2}^k c_u^{drv} \delta_u^{e_{i-1} e_i} + c_u^{drv} \delta_u^{e_k d_u} && \text{(driving)} \\ &+ \sum_{i=1}^k c_u^{svc} \delta_{e_i} && \text{(service)} \\ &+ c_u^{ot} \max\{t_R^{d_u} - t_u^{end}, 0\} && \text{(overtime)} \\ &+ \sum_{i=1}^k \text{LateCost}(e_i, t_R^{e_i}) && \text{(lateness)} \end{aligned}$$

Denoting by  $a_{R_e}$  a binary coefficient with  $a_{R_e} = 1$  if and only if event  $e$  is served in tour  $R$ , the VDP can now be formulated as the following set partitioning problem:

$$\begin{aligned} \text{(IP)} \quad & \min \sum_{R \in \mathcal{R}} c_R x_R \\ & \text{subject to} \quad \sum_{R \in \mathcal{R}} a_{R_e} x_R = 1 \quad \forall e \in E; \\ & \quad \quad \quad \sum_{R \in \mathcal{R}_u} x_R = 1 \quad \forall u \in U; \\ & \quad \quad \quad x_R \in \{0, 1\} \quad \forall R \in \mathcal{R}; \end{aligned}$$

This model describes a dispatch such that every request  $e \in E$  is served exactly once (first constraint), and which contains exactly one tour for each unit  $u \in U$  (second constraint). The objective is the total cost of the tours selected.

**The General Algorithmic Scheme.** Since the number of possible tours is exponential in the number of requests and units, it is evident that not all columns of the coefficient matrix of (IP) can be practically enumerated for our problem size which in our application contains up to 300 requests and 100 units. A standard approach is to use dynamic column generation, which means that we start with a subset  $\tilde{\mathcal{R}} \subset \mathcal{R}$  consisting of a heuristically obtained set of tours forming a feasible solution of (IP). We repeatedly solve (for growing  $\mathcal{R}$ ) a restricted primal LP

$$\begin{aligned} \text{(RLP)} \quad & \min \sum_{R \in \tilde{\mathcal{R}}} c_R x_R \\ & \text{subject to} \quad \sum_{R \in \tilde{\mathcal{R}}} a_{R_e} x_R = 1 \quad \forall e \in E; \\ & \quad \quad \quad \sum_{R \in \tilde{\mathcal{R}}_u} x_R = 1 \quad \forall u \in U; \\ & \quad \quad \quad x_R \geq 0 \quad \forall R \in \tilde{\mathcal{R}}; \end{aligned}$$

whose dual is given by

$$\begin{aligned} \text{(DRLP)} \quad & \max \sum_{e \in E} \pi_e + \sum_{u \in U} \pi_u \\ & \text{s.t.} \quad \sum_{e \in E} a_{R_e} \pi_e + \pi_u \leq c_R \quad \forall R \in \tilde{\mathcal{R}}_u, \forall u \in U; \end{aligned}$$

In the following we briefly describe the general outline of a column-generation algorithm for the VDP which forms the basis of the implementation ZIBDIP (Krumke et al., 2002; Hiller et al., 2004), which has been developed in order to solve (IP) by solving its linear relaxation (LP). Our contribution which is described in detail in Section 1.3 is a new pruning scheme based on computing minimum weight matchings for speeding up the column generation process.

Suppose that we have a currently chosen subset  $\tilde{\mathcal{R}} \subset \mathcal{R}$  of the possible tours in the (LP) and an optimal solution  $\tilde{x}$  of the restricted primal (RLP). For a unit  $u \in U$  and a tour  $R \in \mathcal{R}_u$  let  $\bar{c}_R$  be the reduced cost which can be computed by

$$\bar{c}_R = c_R - \sum_{e \in E} a_{Re} \pi_e - \pi_u.$$

The current optimal solution  $\tilde{x}$  for the restricted primal (RLP) is optimal for the complete primal (LP) if  $\bar{c}_R \geq 0$  for all tours  $R \in \bigcup_{u \in U} \mathcal{R}_u \setminus \tilde{\mathcal{R}}_u$ . (Chvátal, 1983). So a tour  $R$  can improve the current solution of (RLP) only if  $\bar{c}_R < 0$ .

In the pricing step, one searches for columns with negative reduced cost and, if such a column is found, it is added to the columns of (RLP). The restricted primal (RLP) is solved again until no improving column can be found.

**Column Generation.** There are several possibilities to find columns with negative reduced costs. The easiest method is the enumeration of all possible tours in a search tree. Each node in this search tree corresponds to a tour starting at the current position of a unit and ending at the position of the last event served by the unit. A tour associated to a node can be completed to a feasible tour by appending the tour from the position of the last event in the node to the unit's home position. The *pre-cost* of a node in the search tree is the reduced cost of the corresponding tour (without returning to the unit's home position and overtime). The *cost* of a node in the search tree is defined as the reduced cost of the corresponding feasible tour (including the costs for returning to the unit's home position and overtime). The *dual prices* for the events and units are taken from the previous run of the LP solver. The root node  $r$  corresponds to the empty tour. Given a node  $v$  in the tree, the children of  $v$  are obtained by appending one event to  $v$  that is not yet in  $v$ . This way all possible tours can be enumerated.

Since a whole enumeration of all possible tours is practically impossible, we do not scan the whole search tree. In fact, for every node within the search tree we only consider the  $d$  most promising children and we do not consider tours serving more than  $l$  requests for some intelligently chosen  $d$  and  $l$ . The  $d$  most promising children of a node  $v$  and the order in which they are processed is given by sorting the children in the order  $<$  of increasing reduced costs of the new node (*greedy*), or increasing primal cost (*primal-greedy*). Since we are especially interested in tours having low reduced costs, we introduce an accep-

tance threshold  $a$  which means that we skip all tours having greater reduced costs than  $a$ . By the setting of the parameters for the *maximal search depth*  $l$ , the *maximal search degree*  $d$ , and the *acceptance threshold*  $a$  the processing time and the quality of the output can be customized.

As even the dynamically adjusted search tree may still contain a lot of useless nodes, we apply a Branch-and-Bound method in order to explore only the important parts of the search tree. For this reason, we skip every subtree having no chance of containing a node with cost smaller than the acceptance threshold. This procedure of skipping a subtree is called *pruning*.

### 3. The New Pruning Scheme

In this section lower bounds for the tours associated with the nodes within a subtree of a branch-and-bound search tree are presented. The quality of this lower bound is important as a good lower bound makes it possible to make a much better pruning and therefore to iterate less nodes. We present two of these lower bounds. The first one has been used until now and delivered good results. The second one works exactly like the first one with the difference that the lateness costs are taken into account in a much more realistic way. As a result only a fraction of the previously examined nodes has to be computed by now. On the other hand, the computation at each node takes more time, since an assignment problem has to be solved at every node within the search-tree. In fact, we use a two-phase pruning scheme, which only computes the new lower bound if the old lower bound was too weak for pruning. In the last section we provide computational results for the new pruning on several real-world instances.

Every node  $v$  of the search tree corresponds to a sequence of requests  $S(v)$  which are served by a unit  $u$  in the given order. Let  $e_v$  be the last event in  $S(v)$ ,  $t_v$  be its completion time,  $l(v)$  be the length of  $S(v)$ . We denote by  $c(v)$  the reduced cost of the tour corresponding to  $S(v)$ . Let  $E_v = \{e | F_e \subseteq F_u \wedge e \notin S(v)\}$  be the set of all requests which can be appended to  $S(v)$  so that the resulting tour is feasible. Let  $l$  be the maximum depth of the search tree and  $d_v := l - l(v)$  be the maximum number of requests still to append to the tour corresponding to  $v$ . Let  $\text{Subtree}(v)$  be the subtree of  $v$  within the search tree and let

$$\text{Low}(v) := \min_{w \in \text{Subtree}(v)} c(w)$$

be the value of the cheapest node within the subtree of  $v$ . This node corresponds to the minimum reduced cost of a tour beginning with prefix  $S(v)$ .

DEFINITION 1 (MINIMUM REDUCED COSTS OF AN EVENT) *Let*

$$\text{MinRedCost}(e_1, e_2, t) := \text{LateCost}(e_2, t + \delta_u^{e_1 e_2}) + c_u^{svc} \delta_{e_2} - \pi_{e_2}$$

be the minimum reduced costs of the event  $e_2$ , if  $e_1$  is being left for  $e_2$  at time  $t$ . These costs only include minimum lateness and service but no driving and overtime costs.

Therefore,  $\text{MinRedCost}(e_v, e, t_v)$  is a lower bound for the reduced costs of  $e$  in the subtree below  $v$ .

When processing node  $v$  one wishes to derive a lower bound for the reduced costs of a tour beginning with  $S(v)$  resp. a node within the subtree of  $v$ . If this lower bound is greater than the acceptance threshold, the subtree contains no node which corresponds to a tour with reduced costs less than the acceptance threshold and therefore the subtree can be pruned. The initial pruning scheme in the ZIBDIP algorithm of (Krumke et al., 2002) used the following bound:

DEFINITION 2 (MINIMUM REDUCED COST OF A NODE IN A SUBTREE)

$$\text{MaximalGain}(v) := c(v) + \left( \min_{E \subseteq E_v, |E| \leq d_v} \sum_{e \in E} \text{MinRedCost}(e_v, e, t_v) \right)$$

$\text{MaximalGain}(v)$  is a lower bound for the reduced costs of a node within the subtree of  $v$ , since  $\text{MaximalGain}(v)$  assumes that all requests can be served directly after  $e_v$ . Thus we get that

$$\text{MaximalGain}(v) \leq \text{Low}(v). \quad (1)$$

In all moderate load situations this lower bound does in fact a good job in pruning the search tree (see Krumke et al., 2002). Moreover, it is easy and fast to compute.

However, taking a closer look at the particular situation at the ADAC reveals that most of the costs arising are lateness costs implemented to assure customers' short waiting times. Therefore, it seems promising to take a closer look on the lateness costs considered by this lower bound and to develop strategies to improve them. The lateness costs in  $\text{MaximalGain}(v)$  are computed according to the assumption that the service unit starts off for serving each single request in  $E$  immediately after serving  $e_v$ . But as a tour consists of requests which have to be served in a certain order, only one request can be attended at the completion time of the last request in  $S(v)$ . The second one has to wait at least for the service-time of the first request, the third request has to wait for the service-time of both of the previous requests extra and so on. Using the lower bound described above the late costs thereby incurred are being disregarded. Therefore, as a general rule, this lower bound is not strong for  $d_v > 1$  due to the fact that a service unit can serve only one request at a time.

In the following we introduce a new lower bound taking the late costs of sequentialization into account. This bound forms the core of our new pruning scheme. Let  $\delta := \min_{e \in E_v} \delta_e$  be the minimum service time of all requests in

$E_v$ . It is trivial to see that  $\delta$  is in fact a lower bound for the additional service time of all requests  $e \in E_v$ . We note that in the application at the ADAC, it takes a service unit at least 15 minutes to serve a customer and therefore here we have  $\delta \geq 15$ .

Since every unit  $u$  can serve only one request immediately, the second request has to wait for at least  $\delta$  time units longer as if it had been served first and so on. Therefore, there are a lot of lateness costs emerging from the sequentialization of the latter requests which are disregarded by just using  $\text{MaximalGain}(v)$ .

We obtain a lower bound for  $\text{Low}(v)$  by assigning up to  $d_v$  different requests from  $E_v$  to the elements of  $T_{v,\delta} := \{t_v, t_v + \delta, t_v + 2\delta, t_v + 3\delta, \dots, t_v + (d_v - 1)\delta\}$ , such that each  $t \in T_{v,\delta}$  is covered not more than once and the sum of the reduced costs arising by leaving  $e_v$  for  $e \in E_v$  at the assigned point in time is minimal. Not every event  $e \in E_v$  can be assigned to a  $t \in T_{v,\delta}$ , since  $|E_v| \neq |T_{v,\delta}|$  in most cases. This leads to the following definition of a new lower bound:

DEFINITION 3 (LOWER BOUND DUE TO IMPROVED PRUNING)

$$\text{ImprovedBound}(v) := c(v) + \min_{\{e_0, e_1, \dots, e_k\} \subseteq E_v} \sum_{i=0}^k \text{MinRedCost}(e_v, e_i, t_v + i\delta),$$

with  $k \leq d_v - 1, e_i \neq e_j \ \forall i \neq j$

Note that if we only assign a subset of  $T_{v,\delta}$ , it suffices to consider the first ones, since for every request  $e$   $\text{MinRedCost}(e_v, e, t)$  is monotonic increasing in  $t$ . Furthermore, no request  $e$  will be assigned to a time  $t$  with  $\text{MinRedCost}(e_v, e, t) \geq 0$ , as the sum can only become smaller by leaving this request out.

THEOREM 4 *We have*

$$\text{MaximalGain}(v) \leq \text{ImprovedBound}(v).$$

**Proof.** Since for every request  $e$  we have that  $\text{MinRedCost}(e_v, e, t)$  is monotonically increasing in  $t$ , we know that for all  $e \in E_v$  and  $i \in \{0, 1, \dots, d_v - 1\}$  it holds that  $\text{MinRedCost}(e_v, e_i, t_v) \leq \text{MinRedCost}(e_v, e_i, t_v + i\delta)$ . Thus,

$$\begin{aligned} \text{MaximalGain}(v) &= c(v) + \min_{E \subseteq E_v, |E| \leq d_v} \sum_{e \in E} \text{MinRedCost}(e_v, e, t_v) \\ &= c(v) + \min_{\{e_0, e_1, \dots, e_k\} \subseteq E_v, k \leq d_v} \sum_{i=0}^k \text{MinRedCost}(e_v, e_i, t_v) \\ &\leq c(v) + \min_{\{e_0, e_1, \dots, e_k\} \subseteq E_v, k \leq d_v} \sum_{i=0}^k \text{MinRedCost}(e_v, e_i, t_v + i\delta) \\ &= \text{ImprovedBound}(v) \end{aligned}$$



This shows the claim.  $\square$

The following theorem shows that the new bound is in fact a valid lower bound on the minimum reduced cost:

THEOREM 5

$$\text{ImprovedBound}(v) \leq \text{Low}(v).$$

**Proof.** For computing  $\text{ImprovedBound}(v)$ , we only consider service costs and late costs of sequentialization. Since these costs which cannot be omitted,  $\text{ImprovedBound}(v)$  is a lower bound for  $\text{Low}(v)$ .  $\square$

Observe that in many cases  $\text{ImprovedBound}(v)$  is a sharp lower bound for  $\text{Low}(v)$ . Let  $E_s$  be the set of requests to be served after  $e_v$  according to the computation of  $\text{ImprovedBound}(v)$ . If

- 1 all requests  $e \in E_s$  to be served after  $e_v$  lie on the service unit's way home from  $e_v$  in the order of  $\text{ImprovedBound}(v)$ 's computation (no extra driving, no extra lateness),
- 2  $\forall e \in E_s : \delta_e = \delta$  (no extra service cost),
- 3  $t_u^{\text{end}} \geq t_v + (|E_s| - 1)\delta$  (shift end time late enough)
- 4 and no waiting time (no extra lateness)

then  $\text{ImprovedBound}(v) = \text{Low}(v)$

## Computing the New Bound

The problem of assigning events to points in time can be modelled as a minimum weight matching problem in a bipartite graph.

We define a bipartite graph  $G = (V, A)$  with  $V = E \dot{\cup} T$ , where  $E$  contains  $|E_v|$  nodes  $e_i$  ( $i = 0, 1, \dots, |E_v| - 1$ ) for each request  $e_i \in E_v$ . The set  $T$  contains  $d_v$  nodes  $t_j$  ( $j = 0, 1, \dots, d_v - 1$ , where  $t_j$  represents the time  $t_v + j\delta$ ). For every  $i < |E_v|, j < d_v$  with  $\text{MinRedCost}(e_v, e_i, t_v + j\delta) < 0$ , there is an arc  $(e_i, t_j)$  with cost  $\text{MinRedCost}(e_v, e_i, t_v + j\delta)$ .

Let  $M_j$  be the minimum cost matching of cardinality  $j$ . Let  $c(M_j) := \sum_{e \in M_j} c(e)$ . Let  $M$  be the optimal matching with  $c(M) = \min_{j=1, \dots, d_v} c(M_j)$ . The improved bound can then be computed as  $\text{ImprovedBound}(v) = c(v) + c(M)$ . The minimum weight matching problem in a bipartite graph can be reduced to a network flow problem (see Ahuja et al., 1993). Although in general, this reduction does not yield the best possible time bounds, it turns out that in our particular case we can nicely exploit the structure of the successive-shorest path algorithm.

We add a new source vertex  $s$  and connect it by arcs of zero cost to all vertices of  $E$ . All arcs have unit capacity. This results in the the bipartite

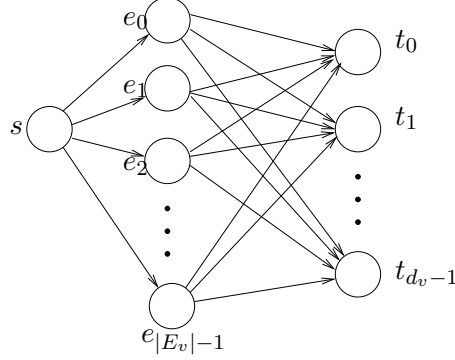


Figure 1. The network corresponding to the assignment problem

graph shown in Figure 1 which we will refer to as *lateness matching network* for brevity.

There is a wide variety of different algorithms which can be used to solve such a minimum cost flow problem. In our case the most promising algorithm is the successive shortest path algorithm. It works by adjusting the imbalances of pairs of sources and sinks by augmenting along paths of minimum weight between them. A detailed description of this algorithm can be found in many books (e.g. Ahuja et al., 1993).

Since we know that there is always an optimal matching of cardinality  $j$ , matching exactly the  $j$  first points in time, we can find  $M_j$  by finding a minimum cost  $b$ -flow with  $b(s) = j$ ,  $b(t_i) = -1, i = 0, \dots, j-1$  and  $b(v) = 0$  for all other nodes.

The minimum cost flow problem in bipartite graphs is solved by applying the following variant of the successive shortest path algorithm.

In Comparison to the standard successive shortest path algorithm, we implement some shortcuts in lines 6, 9, 13 and 16. In the following we will explain their meaning.

Therefore, we need the following propositions:

**THEOREM 6** *Let  $P_j$  and  $P_{j+1}$  be the augmenting paths found by the algorithm ImprovedBound( $v$ ) in the  $j$ th resp.  $j+1$ th. iteration. Then*

$$c(P_j) \leq c(P_{j+1})$$

**Proof.**

Let  $f_j$  be the flow corresponding to  $M_j$  and let  $G_{f_j}$  be its residual graph.  $P_j$  and  $P_{j+1}$  are the augmenting paths of iteration  $j$  resp.  $j+1$  (see Figure 2

---

**Algorithm 1** The Algorithm for the computation of  $\text{ImprovedBound}(v)$ 


---

IMPROVED BOUND

**Input:** An acceptance threshold *threshold*, a *lateness matching network*  
 $G = (V, A)$

**Output:** *true*, if a lower bound *bound* with  $\text{bound} > \text{threshold}$  could be found,  
*false*, otherwise

```

1 Set  $f(e) := 0$  for each  $e \in A$ 
2  $j := 1$ 
3 while  $j \leq d_v$  do
4   Find  $s - t_{j-1}$ -path  $P_j$  in  $G_f$  of minimum weight.
5   if there exists no such path then
6     return true
7   else
8     if  $c(P_j) \geq 0$  then
9       return true
10    Augment  $f$  along  $P_j$ 
11    Let  $M_j$  be the matching corresponding to  $f$ 
12    if  $c(M_j) \leq \text{threshold}$  then
13      return false
14     $\text{bound} := c(M_j) + (d_v - j)c(P_j)$ 
15    if  $\text{bound} > \text{threshold}$  then
16      return true
17     $j := j + 1$ 
18 return true

```

---

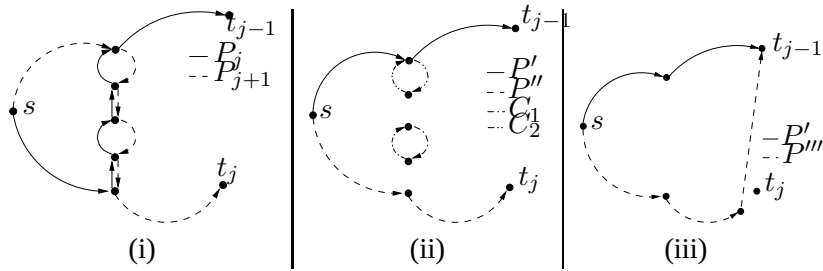


Figure 2. Paths and Cycles in  $G_{f_{j-1}} \cup G_{f_j}$  for the proof of theorem 6

(i)). By deleting all pairs of reverse edges of  $P_j$  and  $P_{j+1}$  we obtain two edge-disjoint cycle-free paths  $P'$  and  $P''$  from  $s$  to  $t_{j-1}$  resp. to  $t_j$  and a set  $\mathcal{C}$  of cycles (see Figure 2 (ii)). Since the reverse edges differ from each other only in their sign, the sum of the costs of a pair of reverse edges is 0 and therefore:

$$c(P_j) + c(P_{j+1}) = c(P') + c(P'') + \sum_{C \in \mathcal{C}} c(C) \quad (2)$$

Since the only difference between  $G_{f_{j-1}}$  and  $G_{f_j}$  is that all edges of  $P_j$  have a different orientation (and of course different costs), every edge in  $P_{j+1}$  which is not part of  $G_{f_{j-1}}$  must have a reverse edge in  $P_j$ . As  $P'$ ,  $P''$  and  $\mathcal{C}$  do not contain any of these edges, they lie in  $G_{f_{j-1}}$ . Since  $f_{j-1}$  is optimal,  $G_{f_{j-1}}$  does not contain any negative cycles and therefore:

$$\begin{aligned} \forall C \in \mathcal{C}: c(C) &\geq 0 \\ \Rightarrow \sum_{C \in \mathcal{C}} c(C) &\geq 0 \end{aligned} \quad (3)$$

Let  $P'''$  be the  $s-t_{j-1}$ -path which is constructed by changing the last edge's target of  $P''$  from  $t_j$  to  $t_{j-1}$  and deleting any cycle which might be gained thereby (see Figure 2 (iii)). If the last edge of  $P'''$  was not part of  $G_{f_{j-1}}$ ,  $P_j$  would contain cycles which is impossible. Hence  $P'''$  is a  $s-t_{j-1}$ -path in  $G_{f_{j-1}}$ . Because of the monotonic increasing reduced cost function and because of the fact that every cycle in  $G_{f_{j-1}}$  is nonnegative:

$$c(P''') \leq c(P'') \quad (4)$$

As  $P_j$  is the shortest  $s-t_{j-1}$ -path in  $G_{f_j}$ :

$$c(P_j) \leq c(P') \quad (5)$$

$$c(P_j) \leq c(P'''). \quad (6)$$

Thus, it follows from above:

$$\begin{aligned}
c(P_j) + c(P_{j+1}) &\stackrel{(2)}{=} c(P') + c(P'') + \sum_{C \in \mathcal{C}} c(C) \\
&\stackrel{(3)}{\geq} c(P') + c(P'') \\
&\stackrel{(4)}{\geq} c(P') + c(P''') \\
&\stackrel{(5),(6)}{\geq} c(P_j) + c(P_{j+1}) \\
&\Rightarrow c(P_j) \leq c(P_{j+1})
\end{aligned}$$

□

**THEOREM 7** *Let  $M_j$  and  $M_i$  be minimum cost matchings of cardinality  $i$  resp.  $j$  with  $i < j$ , then*

$$c(M_j) \geq c(M_i) + (j - i)c(P_i)$$

**Proof.**

$$\begin{aligned}
c(M_j) &= c(M_i) + c(P_{i+1}) + c(P_{i+2}) + \dots + c(P_j) \\
&\stackrel{Thm.6}{\geq} c(M_i) + (j - i)c(P_i)
\end{aligned}$$

□

**THEOREM 8** *Let  $M_k$  be the minimum cost matching and let  $h$  be the highest cardinality of a matching possible among the matchings of  $G$ . Then*

$$c(M_1) \geq c(M_2) \geq \dots \geq c(M_k) \leq c(M_{k+1}) \leq \dots \leq c(M_h)$$

**Proof.** Since  $M_k$  is optimal we know that

$$c(M_{k-1}) \geq c(M_k) = c(M_{k-1}) + c(P_k)$$

and, therefore,  $c(P_k) \leq 0$ . From theorem 6 it follows that for all  $j$  with  $1 \leq j < k$  that  $c(P_j) \leq c(P_k) \leq 0$  and, thus,  $c(M_j) \geq c(M_{j+1})$ . Analogously since  $M_k$  is optimal we know that

$$c(M_k) \leq c(M_{k+1}) = c(M_k) + c(P_{k+1})$$

and, therefore,  $c(P_{k+1}) \geq 0$ . From theorem 6 it follows that for all  $j$  with  $k < j \leq d_v$  we have  $c(P_j) \geq c(P_k) \geq 0$  and, thus,  $c(M_{j-1}) \leq c(M_j)$ . □

As a consequence of the propositions mentioned above, we can justify the shortcuts implemented in the Improved Bound algorithm:

The algorithm can be interrupted in line 12, since we have found a matching  $M_j$  with  $c(M_j) < \text{threshold}$  and, therefore,  $c(M) < \text{threshold}$ .

If  $c(P_j) \geq 0$ , then due to theorem 8 all matchings with a cardinality greater than  $j - 1$  will have higher costs than  $M_{j-1}$ . Since the algorithm has not been interrupted yet by the condition in line 12, for all  $i < j$  we have  $c(M_i) > \text{threshold}$ . This results in  $c(M_j) > \text{threshold}$  for all  $j \leq d_v$  and, therefore, the algorithm can be stopped in line 9.

The algorithm can be terminated in line 6, because it is not possible to find a matching with cardinality  $j$  or higher. That means, the minimum cost matchings of every cardinality have already been computed and since their costs have all been greater than  $\text{threshold}$  we have found our searched lower bound in  $\min_{i \leq j-1} c(M_i)$ . Due to theorem 8 we know that  $M_{j-1}$  is such a matching of minimum cost.

The algorithm can be interrupted in line 14, since for all  $i \geq j : c(M_i) \geq c(M_j) + (i - j)c(P_j) > \text{threshold}$ . And since for all  $i \leq j : c(M_i) > \text{threshold}$ , we know that  $c(M) > \text{threshold}$  and the algorithm can be stopped.

## Computational aspects

Let  $l' := \min_{e \in A} c(e)$  be the cost of the cheapest service possible. Since every lower bound computed by our algorithm estimates the value of a matching of cardinality  $d_v$ , by adding  $-l'$  to each of the arcs from  $E$  to  $T$ , the matchings do not change, but the values of all estimates change by  $d_v l'$ . By doing so we ensure that all arcs have nonnegative costs. Instead of working on the network introduced above we use this slightly modified network. Thus, we can use the DIJKSTRA-algorithm for the first shortest path computation. By using node potentials (see Ahuja et al., 1993), we can use the DIJKSTRA-algorithm for the other iterations, too.

**THEOREM 9** *The running time is  $O(|E_v|^2 d_v)$ .*

**Proof.** We have got to find at most  $d_v$  shortest paths. Having  $n$  nodes, the running time of the DIJKSTRA-algorithm is  $O(n^2)$  (see Korte and Vygen, 2000). Since the number of nodes is  $|E_v| + d_v + 1$  we have got a worst case running time of  $O((|E_v| + d_v + 1)^2 * d_v) = O(|E_v|^2 d_v)$ .  $\square$

## Preprocessing

Since the running time is highly dependent on the number of events considered, we should think about a smaller subset of  $E_v$ .

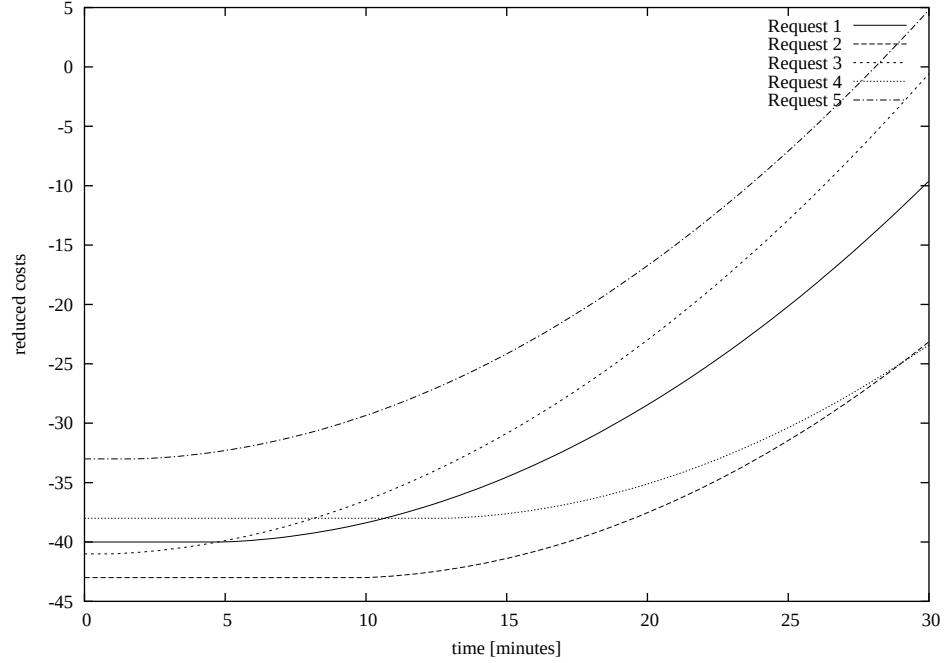


Figure 3. Some reduced cost functions

DEFINITION 10 (K CHEAPEST REQUESTS AT T) *Let*

$$E_{v,t,k} := \{e \in E_v : |\{e' \in E_v : c(e', t) < c(e, t)\}| < k\}$$

*be the set of the  $k$  cheapest requests at  $t$ .*

As  $E_{v,t,k}$  changes with increasing  $t$  (see Figure 3), we need another definition:

DEFINITION 11 (SET OF THE K CHEAPEST REQUESTS IN T) *Let  $E_{v,T,k} := \cup_{t \in T} E_{v,t,k}$  be the set of requests which belong to the  $k$  cheapest requests for at least one  $t \in T$ .*

THEOREM 12 *There is always an optimal matching  $M$  with  $M \subseteq \delta(E_{v,T,d_v})$ .*

**Proof.** Suppose there is a minimal matching  $M$  with  $M \not\subseteq \delta(E_{v,T,d_v})$ . Then there is a node  $w \in E_v \setminus E_{v,T,d_v}$  with  $(w, t_i) \in M, t_i \in T$ . This node does not belong to the  $d_v$  cheapest nodes at time  $t_i$ . As  $M$  is of cardinality at most  $d_v$  and  $|E_{v,t_i,d_v}| \geq d_v$ , there is a node  $x \in E_{v,t_i,d_v}$  which is not incident

to any edge in  $M$ . From the definition of  $E_{v,t_i,d_v}$  follows that  $c((x, t_i)) < c((w, t_i))$  and, therefore, that  $c(M \cup (x, t_i) \setminus (w, t_i)) < c(M)$  which leads to a contradiction.  $\square$

Usually  $E_{v,T,d_v}$  is much smaller than  $E_v$ . Therefore, the cost of computation can be lowered by identifying  $E_{v,T,d_v}$ . Since it takes a worst case running time of  $O(|E_v|^2)$  to determine  $E_{v,T,d_v}$ , we used a superset of  $E_{v,T,d_v}$  which can be determined in linear time:

**DEFINITION 13** *Let  $E'_{v,T,k}$  be the set of requests which belong to the  $k$  cheapest requests at the first point in time of  $T$  and all the requests which are cheaper than at least one of them at the last point in time of  $T$ .*

$$E'_{v,T,k} := E_{v,t_0,k} \bigcup \{e \in E_v \mid \exists e' \in E_{v,t_0,k} : c(e, t_{|T|-1}) < c(e', t_{|T|-1})\}$$

**THEOREM 14**  $E_{v,T,d_v} \subseteq E'_{v,T,d_v}$

**Proof.** Choose  $e \in E_{v,T,d_v}$  arbitrarily. There is a  $t_j \in T$  with  $e \in E_{v,t_j,d_v}$ . If  $j = 0$ , which means that  $e$  belongs to the  $d_v$  cheapest requests at time  $t_0$ , then  $e \in E'_{v,T,d_v}$ . If  $j > 0$  there is an event  $e' \in E_{v,t_0,d_v}$  with  $c(e, t_0) > c(e', t_0)$  and  $c(e, t_j) \leq c(e', t_j)$ . As the requests' reduced cost functions are continuous, there has to be a point of intersection in  $[t_0, t_j]$  of the cost functions of  $e$  and  $e'$ . Because of their specific form there can be no other point of intersection in  $(t_j, t_{d_v-1}]$  and, therefore,  $c(e, t_{d_v-1}) < c(e', t_{d_v-1})$  and  $e \in E'_{v,T,d_v}$ .  $\square$

Since the elements of  $E_v \setminus E'_{v,T,d_v}$  won't be assigned to any element of  $T$ , we can do all the computations introduced in the previous section with  $E'_{v,T,d_v}$  instead of  $E_v$ .

## 4. Experimental Results

In the following, we discuss how well our improved pruning scheme works on real-world instances. Since the new bound is always stronger than the simple pruning using  $\text{MaximalGain}(v)$ , the number of nodes explored is never greater than in the old pruning scheme. On the other hand, the computation of the new bound takes more time. Thus it is not clear upfront whether the new scheme actually results in a speedup.

The test-instances used in our experiments were taken from productive data at ADAC. The computations were carried out on a 3.06 Ghz Pentium 4 machine, equipped with 2 GB of RAM running under Linux (kernel version 2.4.21). We used the GNU C++-compiler gcc, version 3.3.1. The compiler flags were  $-g -O6$ . For solving the linear programs and the final integer linear program, CPLEX version 8.000 was employed.

Table 1 gives an overview of the data used and the computational results. The data considered comprises instances with higher load and lower load,



where the load is defined as  $r := \frac{|R|}{|U|}$ , that is, the ratio of the number of requests and units.

The first column in Table 1 shows the name of the instance, the second one the amount of iterations  $|I|$  computed, the third one the number of requests  $|R|$ , and the fourth one the number of units  $|U|$ . The fifth column contains the amount  $T_{imp}$  of time taken for the column generation in seconds. The sixth column contains the number  $N_{imp}$  of nodes explored in the branch-and-bound search trees by the column generation. The seventh and eighth columns contain analogously the times  $T_{usu}$  and number of nodes  $N_{usu}$  for the column generation worked with the old pruning scheme. In the final two columns we calculated the improvement ratios in order to show how much time has been saved and how much nodes had to be explored by our approach.

| Name          | $ I $ | $ R $ | $ U $ | $T_{imp}$ | $N_{imp}$ | $T_{usu}$ | $N_{usu}$ | $\frac{T_{imp}}{T_{usu}}$ | $\frac{N_{imp}}{N_{usu}}$ |
|---------------|-------|-------|-------|-----------|-----------|-----------|-----------|---------------------------|---------------------------|
| snapshot.240  | 9     | 123   | 100   | 0.74      | 9015      | 0.8       | 11682     | 0.925                     | 0.772                     |
| snapshot.270  | 9     | 125   | 100   | 0.69      | 8927      | 0.81      | 11875     | 0.852                     | 0.752                     |
| snapshot.300  | 9     | 142   | 100   | 1.11      | 12671     | 1.22      | 16976     | 0.91                      | 0.746                     |
| snapshot.330  | 9     | 146   | 100   | 1.09      | 12018     | 1.25      | 16997     | 0.872                     | 0.707                     |
| snapshot.360  | 14    | 156   | 100   | 2.13      | 19974     | 2.59      | 37584     | 0.822                     | 0.531                     |
| snapshot.390  | 9     | 175   | 100   | 2.04      | 17939     | 2.45      | 26376     | 0.833                     | 0.68                      |
| snapshot.420  | 9     | 183   | 100   | 1.75      | 14269     | 2.04      | 21774     | 0.858                     | 0.655                     |
| snapshot.450  | 18    | 200   | 100   | 3.49      | 27267     | 4.36      | 52777     | 0.8                       | 0.517                     |
| snapshot.480  | 12    | 203   | 100   | 2.86      | 19191     | 3.38      | 29961     | 0.846                     | 0.641                     |
| snapshot.510  | 18    | 215   | 100   | 4.35      | 31546     | 5.78      | 65186     | 0.753                     | 0.484                     |
| snapshot.540  | 9     | 216   | 113   | 3.01      | 26029     | 3.49      | 34342     | 0.862                     | 0.758                     |
| snapshot.570  | 9     | 218   | 113   | 2.23      | 16530     | 2.71      | 25386     | 0.823                     | 0.651                     |
| snapshot.600  | 63    | 237   | 113   | 44.35     | 208755    | 120.12    | 932008    | 0.369                     | 0.224                     |
| snapshot.630  | 63    | 250   | 113   | 52.43     | 222860    | 137.87    | 1012705   | 0.38                      | 0.22                      |
| snapshot.660  | 52    | 247   | 113   | 35.61     | 163249    | 82.7      | 616054    | 0.431                     | 0.265                     |
| snapshot.690  | 63    | 261   | 113   | 67.96     | 253480    | 187.9     | 1350234   | 0.362                     | 0.188                     |
| snapshot.720  | 63    | 269   | 113   | 79.18     | 282461    | 249.28    | 1702267   | 0.318                     | 0.166                     |
| snapshot.750  | 63    | 278   | 113   | 90.23     | 301210    | 339.73    | 2247257   | 0.266                     | 0.134                     |
| snapshot.780  | 88    | 291   | 113   | 171.94    | 589261    | 664       | 4395655   | 0.259                     | 0.134                     |
| snapshot.810  | 18    | 290   | 113   | 16.89     | 68629     | 23.95     | 160914    | 0.705                     | 0.426                     |
| snapshot.840  | 33    | 296   | 113   | 41.64     | 170170    | 83.15     | 665072    | 0.501                     | 0.256                     |
| snapshot.870  | 33    | 299   | 113   | 40.59     | 156729    | 65.94     | 569405    | 0.616                     | 0.275                     |
| snapshot.900  | 33    | 308   | 118   | 40.14     | 162978    | 78.74     | 671195    | 0.51                      | 0.243                     |
| snapshot.930  | 25    | 307   | 118   | 33.43     | 147367    | 49.23     | 349847    | 0.679                     | 0.421                     |
| snapshot.960  | 33    | 306   | 118   | 47.07     | 182726    | 81.76     | 645013    | 0.576                     | 0.283                     |
| snapshot.1020 | 33    | 314   | 135   | 36.5      | 150897    | 62.97     | 527625    | 0.58                      | 0.286                     |
| snapshot.1050 | 75    | 322   | 135   | 170.95    | 597153    | 599.28    | 3847145   | 0.285                     | 0.155                     |
| snapshot.1080 | 75    | 311   | 135   | 139.08    | 507560    | 419.07    | 2567749   | 0.332                     | 0.198                     |
| snapshot.1110 | 75    | 316   | 143   | 151.4     | 565927    | 471.38    | 2870062   | 0.321                     | 0.197                     |
| snapshot.1140 | 33    | 314   | 143   | 35.9      | 184356    | 59.13     | 533537    | 0.607                     | 0.346                     |
| snapshot.1170 | 88    | 318   | 143   | 250.96    | 1013204   | 790.36    | 4978175   | 0.318                     | 0.204                     |
| snapshot.1200 | 25    | 308   | 143   | 26.24     | 144604    | 32.7      | 304823    | 0.802                     | 0.474                     |
| snapshot.1230 | 18    | 304   | 143   | 15.44     | 85954     | 18.05     | 167503    | 0.855                     | 0.513                     |
| snapshot.1260 | 25    | 302   | 143   | 22.65     | 148705    | 29.02     | 305682    | 0.78                      | 0.486                     |
| snapshot.1290 | 33    | 292   | 143   | 40.7      | 226991    | 85.43     | 733051    | 0.476                     | 0.31                      |
| snapshot.1320 | 18    | 293   | 143   | 11.1      | 80210     | 13.72     | 166473    | 0.809                     | 0.482                     |
| snapshot.1350 | 9     | 282   | 143   | 5.23      | 41382     | 5.6       | 54662     | 0.934                     | 0.757                     |
| snapshot.1380 | 14    | 281   | 143   | 7.55      | 63766     | 8.63      | 98166     | 0.875                     | 0.65                      |
| snapshot.1410 | 18    | 291   | 143   | 8.71      | 65916     | 9.98      | 113042    | 0.873                     | 0.583                     |
| snapshot.1440 | 12    | 292   | 143   | 5.62      | 34374     | 6.63      | 48744     | 0.848                     | 0.705                     |
| snapshot.1470 | 12    | 293   | 143   | 5.17      | 35067     | 5.92      | 50668     | 0.873                     | 0.692                     |
| snapshot.1500 | 14    | 276   | 144   | 5.36      | 47808     | 6.47      | 82762     | 0.828                     | 0.578                     |
| snapshot.1530 | 9     | 273   | 144   | 3.87      | 30655     | 4.12      | 40840     | 0.939                     | 0.751                     |
| snapshot.1560 | 9     | 273   | 144   | 3.12      | 31857     | 3.3       | 43769     | 0.945                     | 0.728                     |
| snapshot.1590 | 9     | 245   | 144   | 2.26      | 24678     | 2.43      | 33712     | 0.93                      | 0.732                     |

| Name          | I  | R   | U   | $T_{imp}$ | $N_{imp}$ | $T_{usu}$ | $N_{usu}$ | $\frac{T_{imp}}{T_{usu}}$ | $\frac{N_{imp}}{N_{usu}}$ |
|---------------|----|-----|-----|-----------|-----------|-----------|-----------|---------------------------|---------------------------|
| snapshot.1620 | 9  | 247 | 144 | 2.31      | 24327     | 2.49      | 32662     | 0.928                     | 0.745                     |
| snapshot.1650 | 14 | 241 | 143 | 4.22      | 47102     | 4.75      | 75390     | 0.888                     | 0.625                     |
| snapshot.1680 | 9  | 207 | 143 | 1.47      | 19227     | 1.49      | 24727     | 0.987                     | 0.778                     |
| snapshot.1710 | 9  | 204 | 141 | 1.65      | 22202     | 1.67      | 28319     | 0.988                     | 0.784                     |
| snapshot.1740 | 9  | 175 | 139 | 1.23      | 20564     | 1.2       | 25261     | 1.025                     | 0.814                     |
| snapshot.1770 | 9  | 171 | 137 | 1.05      | 16604     | 0.96      | 20996     | 1.094                     | 0.791                     |
| snapshot.1800 | 9  | 158 | 136 | 1.01      | 17134     | 0.96      | 21304     | 1.052                     | 0.804                     |
| snapshot.1830 | 9  | 151 | 132 | 0.73      | 12734     | 0.74      | 16674     | 0.986                     | 0.764                     |
| snapshot.1860 | 9  | 142 | 127 | 0.9       | 15992     | 0.87      | 21011     | 1.034                     | 0.761                     |
| snapshot.1890 | 14 | 135 | 122 | 1.31      | 23321     | 1.35      | 40421     | 0.97                      | 0.577                     |
| snapshot.1920 | 14 | 129 | 119 | 1.63      | 33235     | 1.52      | 54125     | 1.072                     | 0.614                     |
| snapshot.1950 | 20 | 126 | 111 | 4.02      | 75510     | 3.98      | 143581    | 1.01                      | 0.526                     |
| snapshot.1980 | 27 | 140 | 108 | 7.72      | 138405    | 8.15      | 270539    | 0.947                     | 0.512                     |
| snapshot.2010 | 20 | 136 | 96  | 5.53      | 81607     | 6.26      | 163178    | 0.883                     | 0.5                       |
| snapshot.2040 | 27 | 133 | 92  | 14.07     | 161867    | 17.71     | 330388    | 0.794                     | 0.49                      |
| snapshot.2070 | 54 | 135 | 90  | 85.08     | 617194    | 157.55    | 2138728   | 0.54                      | 0.289                     |
| snapshot.2100 | 44 | 135 | 81  | 58.85     | 460058    | 91.59     | 1311138   | 0.643                     | 0.351                     |
| snapshot.2130 | 54 | 135 | 78  | 117.82    | 976590    | 227.24    | 3519523   | 0.518                     | 0.277                     |
| $\Sigma$      |    |     |     | 2045.36   | 9969988   | 5359.92   | 41474697  | 0.382                     | 0.24                      |

Table 1: The performance of improved pruning solving snapshots

It can be observed that in most of the cases, the computation using the new scheme took less time applying the new pruning scheme. Overall, it took only 38.2% of the time and only 24.0% of the nodes were explored compared to the old pruning.

We also examined the dependency of the improvement rate on the maximal search degree and on the maximal search depth of an iteration. Therefore we took all iterations calculated for the computation of the snapshots presented in Table 1, computed their specific improvement rate with respect to the time. The average of these improvement rates with respect to the time for iterations having the same maximal degree and the same maximal depth are illustrated in Figure 4.

This figure shows that improved pruning scheme does always accelerate the column generation and that the performance ratio improves with increasing search degree and increasing search depth of the search tree.

## 5. Conclusions

For the real-world instances considered, the performance ratio due to the improved pruning scheme increased with increasing maximal depth and maximal degree of the search tree. Computing snapshots, it seems always advisable to use our new pruning scheme. Since our pruning scheme estimates the lateness costs more precisely at the cost of a higher computational demand, it has a particularly good performance on instances with high lateness costs.

## References

Ahuja, Ravindra K., Magnanti, Thomas L., and Orlin, James B. (1993). *Network Flows*. Prentice Hall.

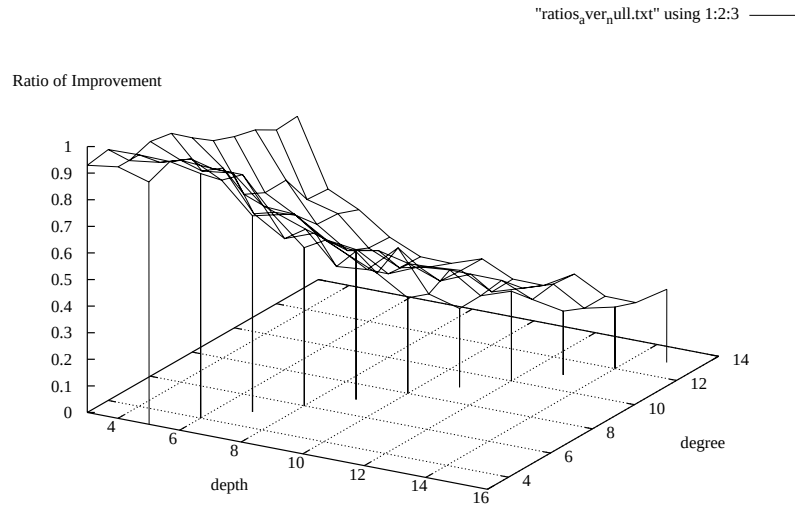


Figure 4. The ratio of the improvement resp. to depth (columns) and degree (rows) of the subtree

- Chvátal, V. (1983). *Linear programming*. A series of books in the mathematical sciences. New York - San Francisco: W. H. Freeman and Company.
- Desrochers, Martin and Desrosiers, Jacques (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354.
- Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). Time constrained routing and scheduling. In Ball, Michael, Magnanti, Tom, Monma, Clyde, and Newhauser, George, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–140. Elsevier, Amsterdam.
- Hiller, B., Krumke, S. O., and Rambau, J. (2004). Reoptimization gaps versus model errors in online-dispatching of service units. In *Proceedings of the Latin-American Conference on Combinatorics, Graphs and Algorithms*.
- Korte, Bernhard and Vygen, Jens (2000). *Combinatorial Optimization*. Springer, Berlin, Heidelberg, New York.
- Krumke, S. O., Rambau, J., and Torres, L. M. (2002). Real-time dispatching of guided and unguided automobile service units with soft time windows. In *Proceedings of the 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 637–648. Springer.

# **Vehicle Routing**

*Abstracts*

# VEHICLE ROUTING AND SCHEDULING WITH TIME VARYING TRAVEL TIMES

Richard Eglese and Will Maden  
*Lancaster University*

**Abstract:** There has been considerable research in vehicle routing and scheduling algorithms and software packages to apply these results in practice. However most of the research published is based on models where the time between nodes on a road network is regarded as fixed. In practice, this is generally not the case and the time taken for any journey may vary significantly by the time of the day, the day of the week and the season of the year in which the journey takes place. This is an issue that may have important consequences for the use of the results from fixed-time models by vehicle fleet managers.

Working with ITIS Holdings plc, a Road Timetable<sup>™</sup> has been created that provides the time and distance between locations for a journey starting at a particular time on a particular day of the week. The data for the Road Timetable<sup>™</sup> has been obtained from a large sample of freight and other vehicles that send details of their position and speed to the ITIS database from which average speeds can be estimated for different roads and different times of day and day of week.

This paper presents a heuristic for the Vehicle Routing Problem with capacity and time constraints that can work with time-varying travel time data such as is provided by the Road Timetable<sup>™</sup>. The heuristic is based on a tabu search algorithm to perform the routing and scheduling.

Results of tests are given using a road network in the north west of England and actual time-varying travel times using Road Timetable<sup>™</sup> information from ITIS Holdings plc. The results demonstrate the potential benefits of routing and scheduling with time-varying travel times, such as the reduction of driver overtime and the reduction of missed service time windows.

**Key words:** Vehicle Routing, Real World Scheduling

# A Dynamic Programming Approach to the Truckload Routing Problem Using a Triplet Formulation

Virginia M. Miori

*Doctoral Candidate, Drexel University*

**Abstract.** Continuous flow truckload routing has always been a challenge. This paper explores the development of continuous flow truckload routes using an alternative formulation of the truckload routing problem (TRP). Rather than viewing the problem as a succession of origin/destination pairs, we look at the problem as a series of routing triplets. We incorporate dynamic programming to solve both the traditional problem formulation and the proposed triplet formulation. We also identify problem characteristics that may be exploited in the triplet formulation, for solution efficiency.

## 1. Introduction

The logistics industry has always provided an abundance of challenges for the field of Operations Research. It has been a continuous source of problems in network design, facility layout, transportation mode selection, and vehicle routing and scheduling among other areas. As technology has progressed, even more opportunities for improvement have been generated. This paper focuses specifically on vehicle routing and scheduling, more specifically the truckload routing problem (TRP).

The importance of the TRP, its extensions and its restrictions can be easily demonstrated by examining our national truckload transportation bill. The term truckload transportation extends an umbrella over various segments of the transportation industry, including but not limited to private fleet operations, truckload common carriers, truckload contract carriers and container movement fleets. According to the Bureau of Transportation Statistics, truckload freight accounted for over \$300 million of the total GDP in the year 2001. It is apparent that even a small improvement in truckload costs would garner impressive reductions in the overall transportation bill.

## 2. Truckload Routing Problem

Truckload operations, no matter the type industry supported, all share several common characteristics. The vehicle (or container) is loaded once with freight that either meets the cubic capacity or weight capacity

of the equipment. All freight loaded is taken to a single destination and completely unloaded at that location.

The practice of truckload routing traditionally begins with a set of nodes which represents all possible lanes (origins/destination pairs) [2]. The nodes are logically viewed in pairs or vehicle movements, one node being categorized as an origin and the other categorized as a destination. The base distribution scenario calls for a loaded movement to be followed by an empty return to the vehicle domicile. In order to defray the cost of the empty return movement, truckload carriers will attempt to pick up an additional truckload on the way back to the domicile. Once the vehicle is back at its domicile, the load assignment process is repeated.

The continuous flow (TRP) attempts to extend this traditional method of operation by building extended routes which combine origin/destination pairs. Rather than returning to its domicile, a vehicle will instead proceed on to pick-up and deliver a subsequent load, which may take it even farther from the domicile. A vehicle stays out on the road for as long as allowable by the Department of Transportation (DOT) hours of service (HOS) rules. For the remainder of this paper, we will use TRP to represent the continuous flow version of the problem.

The TRP has the advantage of creating consecutive loaded movements while maintaining consistency in the route building approach. The movements are chained together into route sequences which typically alternate between loaded and empty movements. The final routes that are created by the TRP may generally be performed at a lower cost than traditional operations and, when displayed on a map, resemble less-than-truckload (LTL) petal routes. We must employ alternate methods than the LTL problem due to many factors, a few of which include: geographic characteristics, nature of time windows and cost characteristics.

This paper supports the emphasis of the TRP on continuous flow routing while providing an alternate approach to the formulation of the TRP. Rather than creating origin/destination pairs, we create node triplets. A node triplet is composed of two origin/destination pairs in succession and may be made up of any two loaded and/or empty movements. This method has been chosen to more effectively consider the opportunity cost of each movement.

A second important aspect of this formulation is that it may apply to either deterministic or stochastic TRP. The stochastic TRP exhibits two sources of uncertainty, demand and cost. The costs associated with the loaded and empty movements vary in accordance with explicit costs such as mileage between the nodes, and both nodes' geographic locations, as well as opportunity costs which result from overall demand

patterns and loads foregone at a specific point in time. We clearly need an effective determination of these costs as applied to each load. Since the triplet notation forces determination of empty movements, these costs are no longer uncertain. Each triplet has a well defined cost structure which considers all of the factors mentioned.

We are then left with a single element of uncertainty, the demand. Demand uncertainty is at the very core of the TRP. Unlike an LTL environment where all demand is known prior to the vehicle route generation, truckload carriers are left with certain knowledge of only the current load assignment. Since a single route may last as long as eight days, vehicles must commence their routes without advanced notice of their subsequent and ultimate direction. We handle this uncertainty through the use of probability distributions.

In this paper we assume a uniform demand probability distribution. Future work will entail detailed examination of demand patterns. In this way, we may more realistically specify demand probability distributions.

### 3. Background

The Traveling Salesman Problem (TSP) has evolved into one of the most studied problems in Operations Research and Computer Science literature. It is also well known that the TSP seeks an optimal solution to a single tour serving  $n$  customers. The Probabilistic Traveling Salesman Problem (PTSP) is presented as a variation on the TSP. Instead of a single tour to be performed once, the PTSP assumes repeated use of a tour, with each instance of the tour requiring a different subset of the customers to be visited. Rather than reoptimizing a TSP for each subset of customers, a single tour of all customers is established. The same tour is followed for each subset of customers, simply eliminating stops at those customers that need not be visited during that instance.

The TSP may be stated more quantitatively as a creation of a single routing tour of  $n$  points. The PTSP may then be restated as a creation of a single routing tour which serves a subset  $k$  of  $n$  points where  $0 \leq k \leq n$ . A probability distribution is used to specify the number and identity of the  $k$  of points to be visited in a given random instance of the problem.

The objective of the PTSP is to find the expected value of the minimum length tour. When solving the PTSP, we must first have a "good" a priori tour through all  $n$  points. This tour must also be good when only  $k$  points are present in a particular instance of the problem.



The PTSP is ideally suited as a basis for the Truckload Transportation Problem (TRP). Like the PTSP, the TRP requires a subset of all customers to be visited on a regular basis. The subset is known a priori on a given day, but is subject to variation and uncertainty in designation of future movements. Due to the additional constraints which we will impose on the PTSP, the TRP may be seen as a restriction of the PTSP.

#### 4. Previous Solution Methods

Many solution methods have been used to solve problems of this nature. These include savings based heuristics (Gronalt et al, 2003) where a relaxed problem formulation, based on network flows, is used to calculate a lower bound to the solution value. Successive linear approximation heuristics (Frantzaskakis et al, 1990) employ rolling horizon procedures to simulate the operation of a truckload carrier.

A new branch and bound approach for solving an integer-programming formulation of the VRP with full truckloads is proposed (Arunapuram et al, 2003). It also considers time windows and waiting costs. The column generation scheme exploits the structure of the problem to solve the linear-programming relaxation problems that arise at the nodes. A 2-path cut to produce better lower bounds and develop an effective separation algorithm allows the incorporation of inequalities into the master problem of a Dantzig-Wolfe decomposition approach (Kohl et al, 1999). The sub-problems are shortest path problems with time window and capacity constraints. Branch and bound is applied to obtain integer solutions to the sub-problems.

Optimal column generation with heuristic variations (Balta et al, 1996) and dynamic assignment (Powell, 1996) have also been applied.

#### 5. TRP Triplet Formulation

We have already noted that the TRP is a restriction of the PTSP. Our discussion now moves to the assumptions and conditions of the TRP since these determine the constraints in the model formulation. The most basic assumption of the TRP is that freight is delivered by truck, in full truckload quantities. This means that a single unit of demand may not be split between multiple vehicles.

We now consider the definition of a route. It is composed of a series of triplets to be serviced by a single vehicle. We must begin and end a route at the same node, known as a domicile. Multiple domiciles may

be present in a single instance of the TRP model, though this paper addresses only one.

Time conditions are placed on the handling of freight by the customers. Loads must be picked up and/or delivered on particular days of the week during specified times of the day. Some customers will be more restrictive than others in specifying their time windows.

The remaining conditions are imposed by the Department of Transportation (DOT). Restrictions are placed on the number of driving hours in a given week and number of work hours in a given week.

As previously stated we utilize the concept of a triplet. Recall that it may be composed of a loaded and empty movement, two loaded movements or two empty movements. The cost of the triplet is therefore the sum of the costs of the individual movements. The transit and work times are the sums of the transit and work times of the individual movements.

One of the primary advantages of the triplet formulation is the elimination of one stochastic element in the TRP. We began with stochastic demand and stochastic cost,

The TRP can be stated as a restricted PSTP. We customize the standard TSP constraints for our triplet formulation and supplement with load, schedule and triplet feasibility, vehicle utilization, and DOT regulations.

The following notation is required for the TRP:

|              |   |                                                  |                                                     |
|--------------|---|--------------------------------------------------|-----------------------------------------------------|
| $N$          | = | the set of all nodes,                            | $1, \dots, N$                                       |
| $V$          | = | the set of all vehicles                          | $1, \dots, V$                                       |
| $ijk$        | = | the set of all triplets,                         | $i=1, \dots, N; j=1, \dots, N \ \& \ k=1, \dots, N$ |
| $c_{ijk}$    | = | cost to service triplet $ijk$                    |                                                     |
| $x_{ijk}^v$  | = | 1 if triplet $ijk$ is served by vehicle $v$      |                                                     |
|              | = | 0 otherwise                                      |                                                     |
| $y_{ij}$     | = | 1 if node pair $ij$ represents a loaded movement |                                                     |
|              | = | 0 otherwise                                      |                                                     |
| $D_i$        | = | departure time from node $i$                     |                                                     |
| $t_i$        | = | travel time between nodes $i$ and $j$            |                                                     |
| $[e_i, l_i]$ | = | early and late arrival times for node $i$        |                                                     |
| $H$          | = | total allowed transit hours per route            |                                                     |
| $w_i$        | = | work hours accumulated at node $i$               |                                                     |
| $W$          | = | total allowed work hours per route               |                                                     |

The objective function may be stated as a cost minimization or a profit maximization. The financial structure of the transportation provider will dictate which of these approaches is preferred. It is likely that a private fleet would choose cost minimization while a for-hire

carrier would select profit maximization. In this paper the formulation is presented as a cost minimization.

$$\min \sum_{ijk} c_{ijk} x_{ijk}^v \quad (1)$$

A routing problem always has specific characteristics that must be modeled, and as such the constraints may be easily categorized into sets. The first set of constraints are for load satisfaction. It guarantees that each lane pair with demand (available load) is serviced at least once. A lane may be served as the first leg of a triplet (origin node to intermediate node) or as the second leg of the triplet (intermediate node to destination node).

$$\sum_{vk} y_{ij}(x_{ijk}^v + x_{kij}^v) = 1 \quad \forall y_{ij} = 1 \quad (2)$$

The load must be carried and additional empty movements may also utilize the same lane. If a lane has no demand, it need not be used in the overall solution. The nature of the optimization ensures that lanes with no demand would only be used and/or reused for empty movements in support of a minimum cost or maximum profit solution. Since we employ a triplet formulation, we must combine lane level information (demand) with triplet level information (decision variables representing inclusion or exclusion of a triplet in the solution) in this constraint set.

The next constraint sets preserves traditional conservation of flow and schedule feasibility. The conservation of flow constraints ensure that every vehicle departs from the final node of every triplet they serve.

$$\sum_{ijlmv} (x_{ijk}^v - x_{klm}^v) = 0 \quad \forall k \quad (3)$$

The schedule feasibility constraints ensures a logical progression through the routes (a vehicle may not depart prior to its arrival), and also forces adherence to time constraints (time windows are satisfied for the pickup and delivery points as well as the domicile). Due to the fact that time windows address individual locations, the time window constraints reflect the individual nodes and not the entire triplet. These are standard constraints used in the TRP.

$$D_i + t_{ij} \leq D_j \quad \forall i, j \quad (4)$$

$$e_i \leq D_i \leq l_i \quad \forall i \quad (5)$$

The remaining constraint set ensures that the routes satisfy the Department of Transportation (DOT) regulations for single drivers. Each

route is restricted to a maximum number of hours in transit. The DOT recently revised these regulations, but the new regulations have been challenged. The model discussed in this paper allows for flexibility in the statement of these constraints to reflect this uncertainty.

$$\sum_{ijk} (t_{ij} + t_{jk}) x_{ijk}^v \leq H \quad \forall v \quad (6)$$

## 6. Lagrangian Relaxation of the TRP

In this section, we present a Lagrangian relaxation of the TRP formulation. This prepares us for the restatement as a Markov Process and the Dynamic Programming formulation.

We introduce a Lagrange multiplier for each constraint to be relaxed and form a linear combination of the constraints with their corresponding Lagrange multipliers as coefficients. Typically the relaxation would include all constraints in the formulation with the intent of reducing the constrained problem to an unconstrained problem.

In this case, only two of the constraints will be relaxed, the DOT hours of service and the load satisfaction constraints. The remaining constraints will be satisfied through later development of transition probabilities in the dynamic programming formulation and solution.

$$\begin{aligned} \min \sum_{ijk} c_{ijk} x_{ijk}^v + \lambda_1 \left( \sum_{ijk} (t_{ij} + t_{jk}) x_{ijk}^v - H \right) \\ + \lambda_2 \left( \sum_{vk} y_{ij} (x_{ijk}^v + x_{kij}^v) - 1 \right) \end{aligned} \quad (7)$$

subject to:

$$\sum_{ijlmv} (x_{ijk}^v - x_{klm}^v) = 0 \quad \forall k \quad (8)$$

$$D_i + t_{ij} \leq D_j \quad \forall i, j \quad (9)$$

$$e_i \leq D_i \leq l_i \quad \forall i \quad (10)$$

The DOT hours of service constraint is weighted by  $\lambda_1$ . The load satisfaction constraint is weighted by  $\lambda_2$ .

## 7. Restatement as Markov Process

The formulation above uses traditional transportation problem notation, which is extended to handle the additional requirements imposed

by the TRP. As noted, a number of solution techniques have been used to address the problem. This paper seeks to solve the TRP using dynamic programming and therefore requires changes in our approach to the problem and in the problem notation.

We move from the traditional view of the problem to a tree structure. The originating node of the tree represents the domicile from which all vehicles commence their routes. The same node also represents the ultimate destination of each route. All successive levels represent the addition of a node<sup>1</sup> to each vehicle's route. Upon selection of a node at each level, examination of the best node to be selected at the next level may take place. The expected depth of any tree will be determined from the average miles accumulated by the nodes<sup>2</sup>.

Dynamic programming solutions depend on our ability to evaluate the cost-to-go of the tree. Given the stochastic nature of demand in the TRP, at best we will be able to calculate the expected cost-to-go. To do so requires probabilistic information. Eventually these probabilities will be determined from a logit model which assesses the likelihood of inclusion of that particular node, at that level, in an optimal solution. Initially, however, we consider random selection which implies equal probability of selection of available nodes at each level.

A key component of the probabilities, no matter the method of determination, is the consideration of only the current level. We do not have to consider the probabilities associated with the selection of nodes in the path up to the current level. This important property, also known as the Markov Property (11), allows us to view the tree as a Finite Markov Process.

$$P(X_{k+1} = x_{k+1} | X_0 = x_0, \dots, X_k = x_k) = P(X_{k+1} = x_{k+1} | X_k = x_k) \quad (11)$$

In our Finite Markov process, transitions between nodes at successive levels are captured in a transition probability matrix. In a homogeneous Markov process, a single stochastic matrix ( $\mathbf{P}$ ) represents transitions at any level. We require instead a time nonhomogeneous Markov Process due to variation in transition probabilities for each level of our tree. Therefore we employ a sequence of stochastic transition probability matrices. The superscript in the notation signifies the level of the dynamic program (tree structure) at which this matrix applies. This deviates from the typical use of the notation.

$$\mathbf{P}^{(k)}, k = 1, \dots, N \quad (12)$$

---

<sup>1</sup> This may be a lane or a triplet.

<sup>2</sup> The total DOT allowable miles divided by the average miles per node.

The determination of the transition probability matrices includes a number of factors, among them is the capture of the time window information. Triplets are served at particular levels of the tree based on the need to deliver loads during the corresponding time frame. Therefore the probability of a triplet being served at a particular level of the tree will increase when the time window matches the level of the tree and decreases otherwise. The inclusion of time window information in the transition probabilities ensures that these constraints will be satisfied. Recall that our relaxed TRP was left with only these constraints resulting in the satisfaction of all constraints in the TRP.

We may now carry the notation and the new view of the problem into a dynamic programming formulation. The transition probability matrices will take a key role in the evaluation of the expected cost-to-go in the dynamic program.

## 8. Dynamic Programming Formulation

Prior to the presentation of the dynamic programming formulation, we must define the necessary functions and notation.

|                          |   |                                                                                                                             |
|--------------------------|---|-----------------------------------------------------------------------------------------------------------------------------|
| $A(k+1,k)$               | = | available nodes at level $k+1$ , depends on node chosen at level $k$                                                        |
| $n_{k+1}$                | = | cardinality of $A(k+1,k)$ (the number of nodes available at level $k+1$ , which depends on the node selected at level $k$ ) |
| $s(k)$                   | = | the node (triplet) chosen at level $k$                                                                                      |
| $\mathbf{P}^{(k)}$       | = | transition probability matrix at level $k$                                                                                  |
| $\mathbf{P}_{i,j}^{(k)}$ | = | transition probability of moving from node $i$ to node $j$ at level $k$                                                     |
| $\mathbf{C}_{s(k),j}$    | = | cost of adding node $j$ after the node chosen at level $k$                                                                  |
| $\mathbf{E}_{q,r}$       | = | cost of moving from stop $q$ to stop $r$                                                                                    |
| $\mathbf{F}_{q,r}$       | = | miles to move from stop $q$ to stop $r$                                                                                     |
| $\mathbf{L}_{q,r}$       | = | loads available between stop $q$ and stop $r$ at level $k$ and node $z$                                                     |
| $T$                      | = | total load requirement for the routing scenario                                                                             |
| $D(k)$                   | = | absolute deviation from the maximum miles allowed by the DOT                                                                |
| $U(i,k)$                 | = | the number of unserved loads if we choose node $i$ at level $k$                                                             |
| $f(i,k)$                 | = | the expected cost to go if we choose node $i$ at stage $k$                                                                  |

|                |   |                                                                                            |
|----------------|---|--------------------------------------------------------------------------------------------|
| $(u,v)$        | = | a lane represented by two vertices; origin and destination                                 |
| $(u,v,w)$      | = | a triplet represented by three vertices: origin, intermediate and destination              |
| $\alpha(s(k))$ | = | the initial stop in the node chosen at (level $k$ $u$ in the triplet representation)       |
| $\beta(s(k))$  | = | the intermediate stop in the node chosen at level $k$ ( $v$ in the triplet representation) |
| $\delta(s(k))$ | = | the final stop in the node chosen at level $k$ ( $w$ in the triplet representation)        |

In the dynamic programming recursion, we are selecting the node  $z$  at the next level,  $\lambda_1$  and  $\lambda_2$ . The node  $z$  minimizes the sum of the cost of that node and the cost-to-go.  $\lambda_1$  and  $\lambda_2$  are Lagrangian multipliers on the penalties; DOT hours of service restriction and load satisfaction. The full recursion follows:

$$f(s(k), k) = \min_{z \in A(k+1, k), \lambda_1, \lambda_2} \{ \mathbf{C}_{s(k), z}^{(k)} + E_{s(k), z} + \lambda_1(D_k) + \lambda_2 U(s(k), k) + \sum_{j \in A(k+2, k+1)} \mathbf{P}_{zj}^{(k)} f(z, k+1) \} \quad (13)$$

The mileage matrix for triplets is generated from the mileage matrix for individual stops. The distance from the initial stop of the triplet to the intermediate stop is added to the mileage from the intermediate stop to the final stop in the triplet. The cost is accumulated in the same fashion. The product of the cost-per-mile for each leg of the triplet, and the mileage for each leg of the triplet is calculated.

$$\mathbf{M}_l = \mathbf{F}_{\alpha(l), \beta(l)} + \mathbf{F}_{\beta(l), \delta(l)} \quad (14)$$

$$\mathbf{C}_l = \mathbf{E}_{\alpha(l), \beta(l)} \cdot \mathbf{F}_{\alpha(l), \beta(l)} + \mathbf{E}_{\beta(l), \delta(l)} \cdot \mathbf{F}_{\beta(l), \delta(l)} \quad (15)$$

Recall that the original formulation of the TRP contains a constraint which reflects the DOT limitation on total miles traveled in a route. This constraint is relaxed and included in the recursion with a penalty term.

$$D(k) = \left| \sum_{g=1}^k [\mathbf{M}_{s(g)} + F_{s(g-1), s(g)}] - m \right| \quad (16)$$

The original formulation also contains a constraint which guarantees service of all loads. An additional term should also be included to penalize unserved loads.

$$U(s(k), k) = \max \{ 0, T - \sum_{g=0}^k [\mathbf{L}_{\alpha(s(g)), \beta(s(g))} + \mathbf{L}_{\beta(s(g)), \delta(s(g))}] \} \quad (17)$$

The connection cost results from discontinuities in the route generated. In the instance of a discontinuity, the cost and distance between the final stop in the departing node and the initial stop in the arriving node is determined for inclusion in the recursion.

$$E_{s(k),z} \quad (18)$$

## 9. Analytic Results

The work presented in this paper is in its early stages. Therefore assumptions were made which removed complexity from the problem. These assumptions focused primarily on the transition probability matrices. The first assumption is that of a constant transition probability matrix, rather than employing a potentially unique matrix at each level.

The second assumption is that of uniformly distributed transition probabilities. At each level of the tree, all available nodes<sup>3</sup> have the same probability of being selected. This assumption has the effect of relaxation of the time window constraints. Triplets are served at particular levels of the tree based on their specified delivery time windows. Therefore the probability of a triplet being served at a particular level of the tree will increase when the time window matches the level of the tree and decrease otherwise.

The following results were created using a sample problem. The problem contains five geographic locations, with one of the locations also acting as the vehicle domicile. Seventeen loads are to be carried in the ultimate routing solution.

Table I demonstrates a typical dispatch solution to the sample problem. The problem is stated in lane notation and you will notice that extended routes are not created. Each route carries a front haul<sup>4</sup> and a few routes also carry a backhaul<sup>5</sup>. One exceptional route carries three loads. The performance measures we use are the total miles traveled, the total cost and the number of vehicles used. We could also calculate average vehicle utilization.

Table II shows a feasible solution generated by the dynamic programming method using triplet notation. As in the typical routing, the routes generated first in succession were more efficient. In fact the first route in this scenario carries six loads. The final few routes all carry

---

<sup>3</sup> The available nodes include any node (triplet) that does not exceed the DOT mileage restriction.

<sup>4</sup> A scheduled loaded movement

<sup>5</sup> The loaded movement is followed by a turn back in the direction of the domicile, along which an additional load may be serviced.



a single load due to the distance that the vehicles must travel to and from the domicile.

Ultimately, Table III shows that the feasible solution generated by the dynamic programming method yielded a savings in miles of over 12%, a savings in cost of nearly 14% and a 30% reduction in the number of vehicles required. The reduced mileage not only contributes to the reduction in route cost, but also contributes to reduced vehicle wear-and-tear. The reduction in vehicles required translates directly into additional cost savings due to fleet size reductions.

## 10. Future Work

An important extension to the results presented in this paper focus on the transition probability matrices. One approach will be accomplished using the logit model often discussed in econometric analysis. The analysis will examine characteristics of the demand, characteristics of the geographic locations and time windows. Eventually, we will be able to view the transition probabilities as decision variables at each level of the optimization.

A second extension is the consideration of the dynamic programming method with an alternate definition of the nodes. Rather than each node representing a triplet, it would instead represent an entire solution to the TRP. Triplets would continue to act as the movement increment for each vehicle. A defining characteristic of the new approach is that it allows the solution to deviate from feasibility where the current solution process does not.

A third direction for future work is the examination of applications within other areas of transportation as well as outside the transportation arena. Any problem area that benefits from a network representation provides a potential area of application.

We look, with anticipation, toward the many avenues for extended research, that have been opened through this new interpretation of the Truckload Routing Problem.

## References

- Arunapuram, Sundararajan, Kamlesh Mathur and Daniel Solow. Vehicle Routing and Scheduling with Full Truckloads. *Transportation Science* Vol. 37, No. 2, May 2003, pp170-182.
- Balta, Rajan, and Mark H. Karwan. The Multiple Vehicle TSP with Time Windows and Equite Constraints over a Multiple Day Horizon. *Transportation Science*, Vol. 30, No. 2, May 1996, pp 120-133.
- Frantaskakis, Lins F., and Warren B. Powell. A Successive Linear Approximation Procedure for Stochastic, Dynamic Vehicle Allocation Problems. *Transportation Science*, Vol. 24, No. 1, Feb. 1990, pp 40-57.
- Gronalt, Manfred, Richard F. Hartl, and Marc Reimann. New savings based algorithms for time constrained pickup and delivery of full truckloads. *European Journal of Operations Research*, 151, 2003, pp. 520-535.
- Kohl, Niklas, Jacques Desrodiers, Oli B.G. Madsen, Marius Solomon, and Francois Soumis. 2-Path Cuts for the Vehicle Routing Problem with Time Windows. *Transportation Science*, Vol. 33, No. 1, Feb. 1999, pp 101-116.
- Powell, Warren B. A Stochastic Formulation of the Dynamic Assignment Problem with an Application to Truckload Motor Carriers. *Transportation Science*, Vol. 30, No. 3, Aug. 1996, pp 195-219.

Table I. Typical Dispatch Solution

| <i>Lane Route</i> | <i>Route Miles</i> | <i>Route Cost</i> |
|-------------------|--------------------|-------------------|
| (1,5) (5,4) (4,1) | 1730               | \$2068.14         |
| (1,4) (4,5) (5,1) | 1730               | \$2060.32         |
| (1,3) (3,2) (2,1) | 1801               | \$2290.24         |
| (1,2) (2,3) (3,1) | 1801               | \$2238.93         |
| (1,2) (2,5) (5,1) | 1818               | \$2341.94         |
| (1,3) (3,4) (4,1) | 3012               | \$3667.86         |
| (1,3) (3,5) (5,1) | 2571               | \$3186.70         |
| (1,4) (4,2) (2,1) | 2426               | \$2965.40         |
| (1,4) (4,3) (3,1) | 3012               | \$3602.10         |
| (1,5) (5,3) (3,1) | 2571               | \$3058.07         |
| Totals            | 22,472             | \$27,480          |

Table II. Triplet Formulation Feasible Solution

| <i>Triplet Route</i>    | <i>Route Miles</i> | <i>Route Cost</i> |
|-------------------------|--------------------|-------------------|
| (1,5,4) (4,5,1) (1,2,1) | 2857               | \$3153.64         |
| (1,1,1) (3,2,3) (3,3,1) | 1972               | \$2060.32         |
| (1,1,1) (1,4,2) (2,5,1) | 3420               | \$2290.24         |
| (1,1,1) (3,4,1) (1,1,1) | 3012               | \$2238.93         |
| (1,1,1) (1,3,5) (5,1,1) | 2571               | \$2341.94         |
| (1,1,1) (1,4,3) (3,1,1) | 2789               | \$3667.86         |
| (1,1,1) (1,5,3) (3,1,1) | 3030               | \$3186.70         |
| Totals                  | 19,651             | \$23,637          |

Table III. Summary Results

|                       |        |
|-----------------------|--------|
| Mileage Reduction     | 2821   |
| –Percentage Savings   | 12.55% |
| Cost Reduction        | \$3843 |
| –Percentage Savings   | 13.98% |
| Reduction in Vehicles | 3      |
| –Percentage Savings   | 30%    |

# **New Trends in Scheduling**

Abstracts

# INVENTORY CONTROL AND SCHEDULING WITH FACILITIES IN PARALLEL

Vladimir Krasik <sup>1</sup>, Joseph Leung <sup>2</sup>, Michael Pinedo <sup>1</sup>, and Jiawei Zhang <sup>1</sup>

<sup>1</sup>*New York University*

<sup>2</sup>*New Jersey Institute of Technology*

Consider a setup with  $m$  identical machines in parallel and  $k$  different product types (a machine may be regarded as equivalent to a facility or a plant). The time needed to produce unit  $j$  of type  $i$  on any one of the machines is  $p_{ij} = p_i$ , i.e., all units of type  $i$  have the same processing time. (If there is only one product type, then the subscript  $i$  is omitted.) Of product type  $i$  there is a demand stream for  $n_i$  units with delivery due dates

$$d_{i1} \leq d_{i2} \leq \dots \leq d_{in_i}.$$

The  $j$ -th unit of product type  $i$  must be delivered by  $d_{ij}$ . The due dates may be regular or irregular. If the due dates for product type  $i$  are regular, then  $d_{i,j+1} - d_{ij}$  is a constant for all  $j \geq 1$ ; this case is the discrete time counterpart of the fixed demand rate assumption that is common in inventory theory. Due dates of product type  $i$  that are regular will be denoted in what follows by  $d_{ij}(\text{reg})$ . The  $j$ -th unit of product type  $i$  is in what follows referred to as job  $(i, j)$ ; it is subject to a penalty cost that is a function of its completion time  $C_{ij}$  and the penalty cost is  $f_{ij}(C_{ij})$ . If  $C_{ij} = d_{ij}$ , then the penalty cost is assumed to be zero, i.e.,  $f_{ij}(d_{ij}) = 0$ . It is assumed that  $f_{ij}$  is increasing for  $C_{ij} \geq d_{ij}$  and decreasing for  $C_{ij} \leq d_{ij}$ . In case of linear costs, product type  $i$  has an earliness cost  $h_i$  per unit time and a tardiness cost  $w_i$  per unit time. If a machine is set up to produce a run (or batch) of products of type  $i$  a setup cost is incurred. Let  $c_{ij}^s$  denote the setup cost incurred just before the processing of the  $j$ th unit of product type  $i$ ; this setup cost is either 0 (i.e., this unit of type  $i$  was immediately preceded by another unit of the same type) or it is  $c_i^s$  (i.e., this unit was preceded by a unit of a different type or the machine had been idle for a while). The objective is to minimize all production costs, including earliness costs, tardiness costs and setup costs.

All cost functions  $f_{ij}(C_{ij})$  considered in this paper satisfy  $f_{ij}(C_{ij}) \geq 0$  for all  $C_{ij}$  and  $f_{ij}(d_{ij}) = 0$ . The following cost functions are of special interest.

(i) (*Linear Cost Functions.*) The function  $f_{ij}(C_{ij})$  is  $w_i(C_{ij} - d_{ij})$  for  $C_{ij} \geq d_{ij}$  and is  $h_i(d_{ij} - C_{ij})$  for  $C_{ij} \leq d_{ij}$ .

(ii) (*Convex Cost Functions.*) The function  $f_{ij}(C_{ij})$  is convex in  $C_{ij}$  and  $f_{ij}(d_{ij}) = 0$ .

(iii) (*Number of Late Jobs.*) The function  $f_{ij}(C_{ij}) = w_i$  for  $C_{ij} > d_{ij}$  and 0 for  $C_{ij} \leq d_{ij}$ .

(iv) (*Fixed Time Windows and Unit Penalties.*) The function  $f_{ij}(C_{ij}) = w_i$  for  $C_{ij} > d_{ij}$  and for  $C_{ij} < d_{ij} - a$ ; the function  $f_{ij}(C_{ij}) = 0$  for  $d_{ij} - a \leq C_{ij} \leq d_{ij}$ .

The modelling framework described above relates to the scheduling and inventory models considered in the literature comprising both similarities and differences between the two classes of models.

The general case with arbitrary  $p_i$  is of course very hard.

We use a notational framework that is a generalization of the classical  $\alpha | \beta | \gamma$  framework used in scheduling theory; the  $\alpha$  refers to the machine environment, the  $\beta$  refers to process restrictions

and constraints, and the  $\gamma$  refers to the objective function. Consider, for example, the problem

$$1 \mid k = 1, p_j = 0, c^s \mid \sum_{j=1}^n (hE_j + wT_j + c_j^s),$$

i.e., there is a single product type, there is a setup cost, and the sum of total earliness, total tardiness and total setup cost has to be minimized. This problem is a discrete time counterpart of a special case of an inventory problem considered by Zangwill (1969). If  $w = \infty$  and the due dates are regular, then the problem reduces to the discrete counterpart of the classical Economic Order Quantity (EOQ) model. If the due dates are not regular and setup costs are constant, then the problem is equivalent to a discrete counterpart of the classical Wagner-Whitin model.

The problems within the framework described above can often be formulated either as a mathematical program or as a dynamic program. Unfortunately, for the former the number of variables and constraints in the various formulations are very large. It can be shown easily that the problem is strongly NP-Hard.

However, there exist for the model described above some properties which lead to polynomial algorithms based on a dynamic programming approach.

**Lemma 1 (Due Date Sequencing Property)** *If the cost function  $f_{ij}(\cdot)$  is convex for each  $i$  and  $j$ , then there exists an optimal schedule in which the jobs of product type  $i$ ,  $i = 1, \dots, k$ , are sequenced in the order of their due dates.*

This Due Date Sequencing Property can be shown to hold for arbitrary  $k$  and  $m$  as long as the cost functions are convex. It can be shown that this Due Date Sequencing Property holds for the parallel machine environment and does not necessarily hold when the cost functions are not convex.

The next preliminary result assumes a formulation of a dynamic program for the single machine, single product case, i.e.,  $m = 1$  and  $k = 1$ . For each pair  $(h, j)$  where  $1 \leq h \leq j \leq n$ , let  $\sigma_{hj}$  denote the optimal sequence assuming we restrict ourselves to the set of active jobs  $h, h+1, \dots, j$ . For  $h > j$ , set  $\sigma_{hj} = \emptyset$ . Let  $V(\sigma_{hj})$  denote the total cost, and  $S_l(\sigma_{hj})$  and  $C_l(\sigma_{hj})$  the starting time and completion time of job  $l$  in sequence  $\sigma_{hj}$ . Consider the optimal schedule  $\sigma_{1n}$ . Let  $j$  be the index of the first job such that  $C_j(\sigma_{1n})$  is strictly less than  $S_{j+1}(\sigma_{1n})$ , i.e., jobs  $1, \dots, j$  are in the first batch with  $j$  being the last job of that batch.

**Lemma 2 (Separation Property)** *If there exists an optimal schedule for the  $n$  jobs such that the last job in a given batch, say job  $j$ , is completed before the starting time of the first job of the next batch (job  $j+1$ ), then the optimal completion time of job  $j$  in the problem that considers only the first  $j$  jobs is strictly less than the optimal starting time of job  $j+1$  in the problem that considers only the last  $n-j$  jobs. That is, if  $C_j(\sigma_{1n}) < S_{j+1}(\sigma_{1n})$ , then  $C_j(\sigma_{1j}) < S_{j+1}(\sigma_{j+1,n})$ .*

In other words, Lemma 2 states the following. The schedule of a busy period in an optimal schedule is entirely independent of the jobs as well as the schedules of the other busy periods. This property does *not* hold when  $k > 1$  or  $m > 1$ .

We have the following preliminary result for the parallel machine problem

$$Pm \mid p_{ij} = p_i, c_i^s \mid \sum_{i=1}^k \sum_{j=1}^{n_i} (f_{ij}(C_{ij}) + c_i^s).$$

**Lemma 3 (One Job Having Zero Cost Property)** *If  $f_{ij}(\cdot)$  is decreasing concave in  $C_{ij}$  for  $C_{ij} \leq d_{ij}$  and increasing concave for  $C_{ij} \geq d_{ij}$ , then there exists an optimal solution in which each busy period has at least one job with a completion time equal to its due date, with the possible exception of the first busy period.*

In the single machine single product case with convex costs, the Separation Property of the optimal schedule allows us to limit our search for an optimal schedule of a set of jobs to combinations of optimal schedules of the disjoint subsets of jobs, leading to a backward dynamic program. In step  $i$  we look for an optimal schedule of the last  $i$  jobs. This property does not hold when  $k > 1$ . Still, the Due Date Sequencing property alone allows us to formulate a dynamic program with the state variables of the value function being the starting times and the number of products to be produced, since given the latter we know that products will be produced in the nondecreasing order of their due dates. The difficulty in this approach is the dimension of the state space of possible starting times, which in the case of an arbitrary convex cost function may not be bounded by a polynomial in the number of jobs. This difficulty can be resolved in the linear costs case, since the One Job Having Zero Cost Property states that this dimension is at most  $O(n^{m(k+1)})$ .

From the above it is clear that the convexity of the cost functions has a major impact on the complexity of the problem. It is to be expected that when cost functions are non-convex, the problems become significantly harder. If the cost function is non-convex, it is very easy to find examples where the Due Date Sequencing Property does not hold.

We also consider a multi-product version of the Number of Late Jobs problem. In this problem, jobs of each product type break down into two subsets: early and late. Within each subset the Due Date Sequencing property holds. Also, in this problem there are no idle periods and, hence, possible starting times are uniquely identified by the total number of jobs. This set of properties allows us to formulate a dynamic program that can be readily extended to the multiple machine case.

We also consider Fixed Time Windows with Unit Penalties. For each job  $j$  there exists a fixed length interval  $[e_j, t_j]$ ; if the job starts after  $e_j$  and completes before  $t_j$ , then the cost is 0, otherwise, the cost will be  $w$ . The cost function is neither convex nor concave. We show that there are two subsets of jobs those scheduled in its interval and those not scheduled in its interval. Within each subset an analog of the Due Date Sequencing property holds. Based on these properties, and one additional concept that involves so-called *auxiliary* jobs, a dynamic program can be formulated.

The table below provides an overview of the time complexities of some of the problems.

|                | Linear<br>Costs | Convex<br>Costs  | Number of<br>Late Jobs | Fixed<br>Timewindows |
|----------------|-----------------|------------------|------------------------|----------------------|
| $m = 1, k = 1$ |                 | $O(n^2 \log(n))$ |                        | $O(n^9)$             |
| $m > 1, k = 1$ |                 | —                |                        | —                    |
| $m = 1, k > 1$ |                 | —                |                        | —                    |
| $m > 1, k > 1$ | $O(n^{mk+m+k})$ | —                | $O(n^{k+m})$           | —                    |

In this paper we establish one more link between inventory theory research and scheduling theory research.

At first, one may wonder why so many of the problems discussed can be solved in polynomial time, while most elementary scheduling problems with earliness and tardiness costs are NP-hard. The reason is clear: in scheduling theory problems with earliness and tardiness costs are NP-hard because they assume there are  $n$  different jobs with different processing times and different cost functions. In our framework, jobs of the same type have the same processing times and cost functions and we consider scheduling problems with a fixed number of types, i.e., we are interested in the special cases with  $k$  being fixed.

## **Others**

Abstracts



# **Analytic Evaluation of the Expectation and Variance of Different Performance Measures for Scheduling Jobs on a Single Machine under Processing Time Variability**

Subhash C. Sarin and Balaji Nagarajan

*Grado Department of Industrial and Systems Engineering  
Virginia Tech, Blacksburg, Virginia, 24061*

**Abstract:** The realm of manufacturing is replete with instances of uncertainties in job processing times, machine statuses (up or down), demand fluctuations, due dates of jobs and job priorities. These uncertainties stem from the inability to gather accurate information about the various parameters (e.g., processing times, product demand) or to gain complete control over the different manufacturing processes that are involved. This uncertainty, or variability, is of considerable importance in the scheduling of production tasks. In many practical cases, a scheduler would prefer to have a stable schedule with minimum variance than a schedule that has lower expected value and unknown (and possibly high) variance. Investigation of the stochastic scheduling literature reveals that the preponderance of the work reported has dealt with optimizing the expected value of the performance measure. By focusing only on the expected value and ignoring the variance of the measure used, the scheduling problem becomes purely deterministic and the significant ramifications of schedule variability are essentially neglected. In light of the above, in this paper, we analyze the single machine scheduling environment and devise algorithms and develop closed-form expressions, wherever possible, to evaluate the expectation and variance of different performance measures for a given schedule. The performance measures considered are both completion time and tardiness based. Processing times are given to be probabilistic and are modeled as random variables with known distributions. The objective of this paper is twofold: i) to comprehensively analyze the single machine scheduling environment by modeling all commonly used performance measures, and ii) to elicit the importance of variability in schedule optimization and the need to determine schedules that are both expectation and variance efficient. The scope of this paper does not include developing solution methodologies to identify expectation-variance schedules. However, our work to evaluate the variance would serve as a precursor to such considerations in stochastic scheduling. The models developed and algorithms devised can be incorporated in various scheduling algorithms that are currently used in practice.

**Key words:** Single machine, stochastic scheduling, processing time variability, variance of a performance measure.

# A Genetic Algorithm with Random Keys Approach for the Job Shop Scheduling Problem with Earliness and Tardiness

*Leonardo Bedoya-Valencia, Elkin Rodriguez-Velásquez, Ghaith Rabadi*  
Old Dominion University, Universidad Nacional de Colombia, Old Dominion University

## Abstract:

The problem addressed in this paper is the Job Shop Scheduling problem when the objective is to minimize the total earliness and tardiness from a common due date for a set of jobs. This objective became very significant after the introduction of the Just-in-time manufacturing approach. The paper presents some preliminary properties of the problem, which are used in addition to a Genetic Algorithm with Random Keys to solve this difficult problem. Finally preliminary results of the approach are reported and some concluding remarks are presented

## Key words:

Job Shop Scheduling, Earliness and Tardiness, Common Due Date, Genetics Algorithm, Random Keys.

# Author Index

- Abdullah S. 413  
Adenso-Diaz B. 93  
Adelson-Velsky G. 597  
Adriaen M. 669  
Alexandridis A. 324  
Angel E. 244  
Armstrong R. 428  
Artigues C. 316  
Aykin T. 515  
Baille F. 585  
Balas E. 2, 520  
Balasubramanian H. 112, 276  
Bampis E. 244, 585  
Bansal N. 518, 655  
Baptiste P. 193, 215  
Barbulescu L. 98  
Barbosa-Povoa A. 46  
Bard J. F. 394  
Bartak R. 552  
Bautista J. 93  
Beliën J. 408  
Bedoya-Valencia L. 745  
Bhattacharya S. 258  
Billaut J-C. 671  
Blazewicz J. 694  
Borne P. 217  
Bose S. K. 258  
Bouibede-Hocine K. 123  
Braeckmans K. 322  
Brucker P. 6, 107, 397  
Bulbul K. 188  
Bunde D. P. 562  
Burda R. 391  
Burke E. 397, 413, 698  
Buscaylet F. 316  
Cai X. 6, 630  
Cepek O. 552  
Chen Z-L. 549  
Cheng P. C-H. 658  
Cheng T.C. E. 694  
Chou S-Y. 299  
Coffman E. 3  
Coppersmith D. 655  
Cucu L. 344  
Dawande M. 326  
De Causmaecker P. 40, 669, 698  
De Maere G. 698  
Degbotse A. 391  
Demeester P. 40  
Demeulemeester E. 322, 335, 341, 408  
Doganis P. 324  
Drezet L-E. 671  
Duenas A. 74  
Duin C.W. 16, 653  
Eglese R. 724  
Erramilli V. 48, 539  
Fan C. 701  
Fayad C. 363  
Feillet D. 316  
Feng G. 196  
Fleischer L. K. 518  
Fong P. Y. 552  
Foong W. 72  
Fordyce K. 391  
Fowler J. 42, 44, 112, 186, 276, 288, 392, 643  
Gao S. 428  
Garibaldi J.M. 238  
Geismar H. N. 326  
Gelbukh A. 597  
Glass C. A. 69  
Gomes M. C. 46  
Gordon V. S. 132  
Gosling T. 485  
Guirchoun S. 232  
Hanen C. 445, 453  
Hall N. 6  
Hendel Y. 140  
Herrmann J. W. 380  
Herroelen W. 213, 322, 335, 341  
Howe A. E. 98  
Huo Y. 573  
Jampani J. 539  
Jouglet A. 215  
Juhn J.H. 340  
Kamel A. E. 217  
Kampmeyer T. 107  
Karaesmen I. Z. 19  
Keha A. 112, 186, 288, 643  
Keskinocak P. 19  
Kelly J. D. 327  
Khowala K. 643  
Kimbrel T. 337, 518  
Krasik V. 740  
Krumke S. O. 705  
Kumar M. 701  
Kutanoglu E. 44, 334  
Kwan R. S. K. 109, 702  
Kwan A. S. K. 109, 702  
Laforest C. 585  
Lambrechts O. 335  
Laplagne I. 109, 702  
Lau H. C. 196  
Lau H. Y. K. 536  
Laub J. D. 288  
Lei L. 70, 428, 701  
Lenstra J. K. 4  
Leung J. Y. T. 471, 501, 516, 573, 740  
Leus R. 213  
Levner E. 597  
Li H. 471  
Lin S-W. 299  
Liu S. 70  
Machowiak M. 694  
Maden W. 724  
Maenhout B. 410  
Mahdian M. 518  
Maier H. R. 72

Maravelias C. T. 66  
 Martineau P. 232  
 Mason S. J. 42, 44, 48, 276, 539  
 McCollum B. 413  
 McKenna J. S. 515  
 Miller T. 519  
 Milne J. 391  
 Miori V. M. 725  
 Monkman S. K. 394  
 Montana D. 353  
 Morrice D. J. 394  
 Morrison J. R. 391  
 Mulder J. 698  
 Nagarajan B. 744  
 Ng D. S.T. 69  
 Nikolakopoulos A. 8  
 Novais A. Q. 46  
 Oguz C. 162, 694  
 Ouelhadj D. 238  
 Owusu G. 485  
 Paelinck M. 698  
 Pan C. 536  
 Pape C. L. 193  
 Park S. 70  
 Pascual F. 244  
 Patrinos P. 324  
 Pereira J. 93  
 Petrovic D. 74, 363  
 Petrovic S. 363  
 Pfund M. 42, 276, 392  
 Pinedo M. 471, 740  
 Piplani R. 552  
 Post G. 397  
 Pundoor G. 549  
 Qu R. 397  
 Ranson D. 658  
 Rapine C. 585  
 Rabadi G. 745  
 Rashidi H. 677  
 Rios Solis Y. A. 169  
 Roberts M. 98  
 Rodriguez-Velásquez E. 745  
 Rudova H. 11  
 Sakellariou R. 238  
 Sarimveis H. 8, 324  
 Sarin S. C. 673, 744  
 Savourey D. 215  
 Schieber B. 518  
 Shabtay D. 149  
 Shchepin E. 606  
 Simonetti N. 520  
 Simpson A. R. 72  
 Smith S. 6  
 Sorel Y. 344  
 Souhkal A. 232  
 Sourd F. 140, 169  
 Sriskandaranjah C. 326  
 Steiner G. 149  
 Sung C. S. 340  
 Svirdenko M. 518, 655  
 Tanrisever F. 334  
 Tarasevich A. A. 132  
 Tayur S. 19  
 T'kindt V. 123  
 Tsang E. P. K. 6, 485, 677  
 Vakhania N. 606  
 Van de Vonder S. 341  
 Van der Sluis H.J. 16, 653  
 Vanden Berghe G. 40, 669  
 Vanhoucke M. 410  
 Varadarajan A. 673  
 Vazacopoulos A. 6, 520  
 Verbeke B. 40  
 Virginas B. 485  
 Vlk M. 11  
 Voudouris C. 485  
 Wang Y. 673  
 Wang Z. 533  
 Westphal S. 705  
 Whitley L. D. 98  
 Williamson E. 391  
 Wong R. C. W. 516  
 Wu X. 630  
 Xing W. 533  
 Yang J. 701  
 Yang W. 19  
 Yoon S. H. 340  
 Yucel E. 162  
 Zhang J. 740  
 Zhao H. 501  
 Zhao Y. 536  
 Zhou X. 630  
 Zinder Y. 445, 453  
 Zribi N. 217