Proceedings of the

3rd Multidisciplinary International Conference on Scheduling: Theory and Application

Paris, France
28 - 31 August 2007

Edited by:

Philippe Baptiste
Graham Kendall
Alix Munier-Kordon
Francis Sourd

Editors

Philippe Baptiste

CNRS LIX
Ecole Polytechnique,
F-91128 Palaiseau,
France.
E-mail: Philippe.Baptiste@polytechnique.fr

Graham Kendall

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

Alix Munier-Kordon

Laboratoire d'Informatique de Paris 6
4, place Jussieu,
75 252 Paris, cedex 05,
France.
E-mail: Alix.Munier@lip6.fr

Francis Sourd

Laboratoire d'Informatique de Paris 6
4, place Jussieu,
75 252 Paris, cedex 05,
France.
E-mail: Francis.Sourd@lip6.fr

Preface

The Third Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007) is being held in Paris (France), August 28-31, 2007 and is hosted by Pierre & MarieCurie University and Ecole Polytechnique.

This conference is the third in a series of conferences (the first took place in Nottingham, UK in August 2003, the second in NewYork in July 2005) that serves as a forum for an international community of researchers, practitioners and vendors on all aspects of multi-disciplinary scheduling. The aim is to bring together scheduling researchers and practitioners from all the disciplines that engage with scheduling research.

The proceedings contain 93 contributions. In section one, you will find 7 invited articles that will be presented during the plenary sessions. In section two, you will find 61 regular papers, and the final section contains 25 extended abstracts. The 86 non-invited contributions have been selected from among 177 submissions by the International Advisory Committee and the Conference Program Committee. We would like to thank the members of both committees for their valuable contribution to the conference.

We would also like to thank Christoph Dürr, Antoine Jouglet and Evelyne Rayssac for their invaluable work in helping us to organize the conference. It would not have been possible without their help.

Philippe Baptiste
Graham Kendall
Alix Munier-Kordon
Francis Sourd

Sponsors

Conference Program Committee

Philippe Baptiste (co-chair)	CNRS LIX, Ecole Polytechnique, Paris, France
Graham Kendall (co-chair)	The University of Nottingham
Alix Munier-Kordon (co-chair)	Université de Paris 6
Francis Sourd (co-chair)	CNRS, Université de Paris 6

Uwe Aickelin,	The University of Nottingham, UK
Hesham Alfares,	King Fahd University of Petroleum & Minerals, Saudi Arabia
Ali Allahverdi,	Kuwait University, Kuwait
Christian Artigues,	LAAS-CNRS, Toulouse, France
Chris Beck,	University of Toronto, Canada
Jacques Carlier,	Université de Technologie, Compiègne, France
Patrick De Causmaecker,	K.U. Leuven Campus Kortrijk, Belgium
Zhi-Long Chen,	University of Maryland, USA
T. C. E. Cheng,	The Hong Kong Polytechnic University, Hong Kong
Philippe Chrétienne,	Université Paris 6, France
Stéphane Dauzère-Pérès,	Ecole des Mines de Saint-Etienne, France
Mauro Dell'Amico,	University of Modena and Reggio Emilia, Italy
Erik Demeulemeester,	K.U.Leuven, Belgium
Kath Dowsland,	University of Nottingham, UK
Maciej Drozdowski,	Poznañ University of Technology, Poland
Christoph Dürr,	CNRS, Ecole Polytechnique, France
Wilhelm Erben,	Fakultät Informatik, Konstanz, Germany
Gerd Finke,	Laboratoire LEIBNIZ, Grenoble
Dalibor Froncek,	University of Minnesota Duluth, USA
Michel Gendreau,	Université de Montréal, Canada
Celia A. Glass,	Cass Business School, City University, London, UK
Alain Guinet,	INSA de Lyon, France
Jin-Kao Hao,	University of Angers, France
Pascal Van Hentenryck,	Brown University, USA
Martin Henz,	National University of Singapore, Singapore
Jeffrey W. Herrmann,	University of Maryland, USA
Willy Herroelen,	Research Center for Operations Management, Leuven
Han Hoogeveen,	Utrecht University, The Netherlands
Adam Janiak,	Wroclaw University of Technology, Poland
Vincent Jost,	Ecole Polytechnique Fédérale de Lauzanne, Suisse
Antoine Jouglet,	Université de Technologie, Compiègne, France
Dr. Jeffrey H. Kingston,	The University of Sydney, Australia
Wieslaw Kubiak,	Memorial University, St. John\'s, Canada
Erhan Kutanoglu,	The University of Texas at Austin, USA
Raymond Kwan,	University of Leeds, UK
Hoong Chuin Lau,	Singapore Management University, Singapore
C. Y. Lee,	Hong Kong University of Science and Technology, Hong Kong
Lei Lei,	Rutgers University, USA
Joseph Leung,	New Jersey Institute of Technology
Eugene Levner,	Universidad de La Laguna, Spain

Arne Løkketangen,	Molde University College, Norway
Dirk Chr. Mattfeld,	TU Braunschweig, Germany
Barry McCollum,	Queen's University, Belfast, UK
Martin Middendorf,	University of Leipzig, Germany
Rolf Möhring,	Technische Universität Berlin, Germany
Bryan A. Norman,	University of Pittsburgh, USA
Wim Nuijten,	ILOG, France
Ceyda Oguz,	Koc University, Istanbul, Turkey
Costas Pappis,	University of Piraeus, Greece
Erwin Pesch,	University of Siegen, Germany
Dobrila Petrovic,	Coventry University
Sanja Petrovic,	University of Nottingham, UK
Marie-Claude Portmann,	Ecole des Mines de Nancy, France
Christian Prins,	University of Technology of Troyes, France
Kirk Pruhs,	University of Pittsburgh, USA
Vic J. Rayward-Smith,	University of East Anglia, UK
Andrea Schaerf,	Università di Udine
Andreas S. Schulz,	Massachusetts Institute of Technology, USA
Roman Slowinski,	Poznañ University of Technology, Poland
Vincent T'kindt,	Université de Tours, France
Jonathan Thompson,	Cardiff University, UK
Denis Trystram,	ID-IMAG, Grenoble, France
Greet Vanden Berghe,	KaHo St.-Lieven, Gent, Belgium
Alkis Vazacopoulos,	Dash Optimization, USA
Jean-Paul Watson,	Sandia National Laboratories, Albuquerque, USA
Joel Wein,	Polytechnic University, New York
Yakov Zinder,	University of Technology, Sydney, Australia

Organizing Committee

Philippe Baptiste (co-chair)	CNRS LIX, Ecole Polytechnique, Paris, France
Alix Munier-Kordon (co-chair)	Université de Paris 6
Francis Sourd (co-chair)	CNRS, Université de Paris 6
Christoph Dürr	CNRS, Ecole Polytechnique, France
Antoine Jouglet	Université de Technologie, Compiègne, France

International Advisory Committee

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

Contents

Part I. Invited Speakers

The Job Shop Problem: Old and New Challenges 15
Peter Brucker

Walk the (Time-)Line: Scheduling and Execution 23
Amedeo Cesta

Scheduling with Batch Compatible Tasks 24
Gerd Finke

Scheduling in Highly Uncertain Environments 27
Rhonda Righter

Interval Scheduling 33
Frits Spieksma

Online Stochastic Routing and Scheduling 41
Pascal Van Hentenryck

Robotics and Chromatic Scheduling 42
Dominique de Werra

Part II. Regular Papers

Semi On-Line Scheduling on Two Uniform Processors 51
Enrico Angelelli, Maria Grazia Speranza and Zsolt Tuza

Determining Rules in Fuzzy Multiple Heuristic Orderings for Constructing Examination Timetables 59
Hiishamuddin Asmuni, Edmund K. Burke, Jonathan M. Garibaldi and Barry McCollum

A Simulated Annealing Hyper-heuristic : Adaptive Heuristic Selection for Different Vehicle Routing Problems 67
Ruibin Bai, Edmund K. Burke, Michel Gendreau and Graham Kendall

Sequencing a Single Machine with Due Dates and Deadlines: an ILP-based Approach to Solve Very Large Instances 71
Philippe Baptiste, Federico Della Croce, Andrea Grosso and Vincent Tkindt

Order Acceptance and Scheduling Decisions in Make-to-Order Systems 80
Zehra Bilgintürk, Ceyda Oğuz and Sibel Salman

Scheduling with Special Case of Multipurpose Machines 88
Mourad Boudhar and Hamza Tchikou

A Parallel Metaheuristics for the Single Machine Total Weighted Tardiness Problem with Sequence-Dependent Setup Times 96
Wojciech Bozejko and Mieczyslaw Wodecki

Optimization of the Departure Schedule at a Public Transit Terminal with Multiple Destinations 104
Giuseppe Bruno, Gennaro Improta and Antonino Sgalambro

A Decomposition Heuristic for Planning and Scheduling of Jobs on Unrelated Parallel Machines 112
Habla Christoph, Lars Mönch, Michele E. Pfund and John W. Fowler

Scheduling Steel Production using Mixed-Integer Programming and Constraint Programming 120
Andrew Davenport and Jayant Kalagnanam

Towards the Optimal Solution of the Multiprocessor Scheduling Problem with Communication Delays 128
Tatjana Davidović, Leo Liberti, Nelson Maculan and Nenad Mladenović

The Cost of Scheduling Customers in Routing Problems 136
Wout Dullaert, Olli Bräysy and Amir Salehipour

Time-Indexed Formulations and a Large Neighborhood Search for the Resource-Constrained Modulo Scheduling Problem 144
Benoit Dupont de Dinechin

Scheduling and Location (ScheLoc): Makespan Problem with Variable Release Dates 152
Donatas Elvikis, Horst W. Hamacher and Marcel T. Kalsch

Memetic Algorithms and Hyperhill-climbers 159
Ersan Ersoy, Ender Özcan and A. Şima Uyar

Applying Optimal Control to Jetty Scheduling Problems 167
Fabio Dias Fagundez and João Lauro Dorneles Facó

A Decision Support System for Scheduling a Painting Facility in an Automotive Supplier 175
Jose Pedro García-Sabater, Carlos Andres, Julio Juan García-Sabater and Cristóbal Miralles

The Two-Stage Assembly Flow Shop Scheduling with an Availability Constraint 184
Hatem Hadda, Najoua Dridi and Sonia Hajri-Gabouj

Distributed Decision Making in Hospital Wide Nurse Rostering Problems 192
Stefaan Haspeslagh, Patrick De Causmaecker and Greet Vanden Berghe

Non-preemptive Scheduling of Distance Constrained Tasks Subject to Minimizing Processor Load 200
Klaus H. Hecker and Alexander Hasenfuss

Medium-Term Production and Staff Planning in the Automotive Industry 209
Claas Hemig and Jürgen Zimmermann

Climbing Depth-bounded Discrepancy Search for Solving Flexible Job Shop Scheduling Problems 217
Abir Ben Hmida, Marie-José Huguet, Pierre Lopez and Mohamed Haouari

Complex Job Shop Multiple Orders per Job Scheduling 225
Jagadish Jampani and Scott J. Mason

New Lower Bound for the Bin Packing Problem 234
Bassem Jarboui and Abdelwaheb Rebaï

Optimization vs. Persistence in Scheduling: Heuristics to Manage Uncertainty in On-Demand Air Travel 242
Itir Z. Karaesmen, Wei Yang and Pinar Keskinocak

Solving Resource-Constrained Project Scheduling Problem with Particle Swarm Optimization 251
Sylverin Kemmoé Tchomté, Michel Gourgand and Alain Quilliot

Analyzing Basic Representation Choices in Oversubscribed Scheduling Problems 259
Laurence A. Kramer, Laura V. Barbulescu and Stephen F. Smith

A Case Study of an Integer Programming Model for Instructor Assignments and Scheduling Problem 267
Serge Kruk and Eddie Cheng

Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems 276
Philippe Laborie and Daniel Godard

A Disjunctive Graph for the Job-Shop with Several Robots 285
Philippe Lacomme, Mohand Larabi and Nikolay Tchernev

A Compact Optimization Approach for Job-Shop Problems 293
Giuseppe Lancia, Franca Rinaldi and Paolo Serafini

Minimizing Makespan with Multiple Orders per Job in Mixed Flowshops 301
Jeffrey D. Laub, John W. Fowler and Ahmet B. Keha

Estimation of Absolute Error for Minimization Maximum Lateness 309
Alexander A. Lazarev

Scheduling the Operations of an Integrated Production-Distribution Process 316
Lei Lei, Wanpracha Art Chaovalitwongse and Selim Bora

Generating Job Schedules for Vessel Operations in a Container Terminal 328
Thin Yin Leong and Hoong Chuin Lau

Scheduling Tests on Vehicle Prototypes using Constraint Programming 336
Kamol Limtanyakul and Uwe Schwiegelshohn

iSchedule, an Optimisation Toolkit for Complex Scheduling 344
Anne Liret, David Lesaint, Raphael Dorne and Chris Voudouris

On a Generalized Graph Coloring/Batch Scheduling Problem 353
Giorgio Lucarelli, Ioannis Milis and Vangelis Th. Paschos

Inventory Routing Problem Solved by Heuristic Based on Column Generation . . 361
Sophie Michel and François Vanderbeck

EDF Feasibility and Hardware Accelerators 368
Andrew Morton and Wayne M. Loucks

An Approximation Algorithm for the UET Two-Machine Open-Shop Problem with Time Delays 377
Alix Munier-Kordon and Djamal Rebaine

A Framework for School Timetabling Problem 386
Kimmo Nurmi and Jari Kyngäs

A Memetic Algorithm for Solving a Timetabling Problem: An Incremental Strategy 394
Ender Özcan and Alpay Alkan

Scheduling in a Multi-Processor Environment with Deteriorating Job Processing Times and Decreasing Values: the Case of Forest Fires 402
Costas Pappis and Nikos Rachaniotis

A Window Time Negotiation Approach at the Scheduling Level inside Supply Chains 410
Marie-Claude Portmann and Zerouk Mouloua

Adaptive Decomposition and Construction for Examination Timetabling Problems 418
Rong Qu and Edmund K. Burke

Aligning Frequencies in Cyclic Delivery Scheduling 426
Birger Raa and El-Houssaine Aghezzaf

Scheduling Single Round Robin Tournaments with Fixed Venues 431
Rafel A. Melo, Sebastián Urrutia and Celso C. Ribeiro

Unrelated Parallel Machines Scheduling with Resource-Assignable Sequence Dependent Setup Times 439
Rubén Ruiz and Carlos Andrés

Minimizing the Total Weighted Number of Late Jobs with Late Deliveries in Two-Level Supply Chains 447
George Steiner and Rui Zhang

Genetic Algorithm for Late Work Minimization in a Flow Shop System 455
Malgorzata Sterna, Jacek Blazewicz and Erwin Pesch

Cyclic Scheduling of Tasks with Unit Processing Time on Dedicated Sets of Parallel Identical Processors 463
Přemysl Šůcha and Zdeněk Hanzálek

Evaluating Online Scheduling Techniques in Uncertain Environments 471
Michael Thomas and Helena Szczerbicka

Time Optimal Periodic Task Scheduling with Storage Requirement Minimization 480
Sid-Ahmed-Ali Touati

Large-Scale Short-Term Planning in Chemical Batch Production 490
Norbert Trautmann and Christoph Schwindt

Single Machine and Parallel Machine Scheduling Problems with a Common Due Date to Minimize Total Weighted Tardiness 498
Nguyen Huynh Tuong, Ameur Soukhal and Jean-Charles Billaut

A Combined Meta-Heuristic with Hyper-Heuristic Approach to the Scheduling of the Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Machines 506
José Antonio Vázquez Rodríguez, Sanja Petrovic and Abdellah Salhi

Competitive Agent Scheduling with Controllable Processing Times 514
Guohua Wan, Joseph Y.-T. Leung and Michael Pinedo

Computing Lower Bounds for the Schedule of a Multifunction Radar 523
Emilie Winter and Ruslan Sadykov

The Strength of Priority Algorithms 531
Yakov Zinder

Two-machine Shop Floor Scheduling Problem with Multi-Predecessor Constraints 538
Xiandong Zhang, Jatinder N.D. Gupta

Part III. Abstracts

Statistical Quality Analysis of Schedulers under Soft-Real-Time Constraints 547
Hilbrandt Baarsma, Johann Hurink and Pierre Jansen

Sequencing JIT Mixed Model Assembly Lines Under Station-Load and Part-Usage Constraints using Lagrangean Relaxations 550
Joaquín Bautista Valhondo and Jordi Pereira Gude

Clustering Within Timetabling Conflict Graphs 553
Camille Beyrouthy, Edmund K. Burke, Barry McCollum, Paul McMullan, Dario Landa-Silva and Andrew John Parkes

Dynamic Cooperative Search for Constraint Satisfaction and Combinatorial Optimization: Application to a Rostering Problem 557
Boris Bontoux, Dominique Feillet, Christian Artigues and Eric Bourreau

Selecting and Scheduling Tasks with Agreeable Time Windows and Setup Costs 561
Philippe Chrétienne and Francis Sourd

An Educational Tool for the Resource-Constrained Project Scheduling Problem 564
Filip Deblaere, Erik L. Demeulemeester and Willy S. Herroelen

Scheduling Resumable Deteriorating Jobs 567
Stanislaw Gawiejnowicz and Alexander Kononov

The Parallel Machine Scheduling Problem with Path-like Constraints 570
Yann Hendel and Wieslaw Kubiak

Dynamic Algorithms for Order Acceptance and Capacity Planning within a Multi-Project Environment 572
Jade Herbots, Willy S. Herroelen, Roel Leus and Erik L. Demeulemeester

Exact and Suboptimal Reactive Strategies for Resource-Constrained Project Scheduling with Uncertain Resource Availabilities 575
Olivier Lambrecht, Erik L. Demeulemeester, Willy S. Herroelen

Integrated Vehicle and Crew Scheduling for Extra-Urban Transport 578
Benoît Laurent and Jin-Kao Hao

Scheduling Problems of Chemical Experiments 581
Vassilissa Lehoux-Lebacque, Nadia Brauner, Gerd Finke and Christophe Rapine

Project Scheduling with Success or Failure in Individual Activities 584
Roel Leus and Bert De Reyck

A Branch-and-Price Procedure for Nurse Staffing Incorporating Roster Preferences 588
Broos Maenhout and Mario Vanhoucke

Interactive Evolutionary Multicriteria Scheduling 591
Jon Marquis, John W. Fowler, Esma Gel, Murat Koksalan, Pekka Korhonen and Jyrki Wallenius

Constructive versus Improvement Heuristics: An Investigation of Examination Timetabling 595
Paul McMullan, Barry McCollum, Edmund K. Burke and Graham Kendall

Real-time Student Sectioning 598
Keith Murray and Tomáš Müller

Staff Rostering Problem: Choice of a Mathematical Model 601
Édith Naudin, Peter Chan, Michael Hiroux, Georges Weil and Tahar Zémmouri

Online Scheduling of Parallel Jobs on Two Machines is 2-Competitive 605
Jacob Jan Paulus and Johann L. Hurink

Multi-skill Project Scheduling Problem and Total Productive Maintenance 608
Cédric Pessan, Odile Bellenguez-Morineau and Emmanuel Néron

A Methodology for Integrated Risk Management and Proactive Scheduling of Construction Projects 611
Damien Schatteman, Willy S. Herroelen, Stijn Van de Vonder and Anton Boone

An Exact Algorithm for Single-Machine Scheduling without Idle Time 614
Shunji Tanaka

An Efficient Heuristic Procedure for the Resource Availability Cost Problem ... 618
Vincent Van Peteghem and Mario Vanhoucke

A Strengthened Continuous Time Formulation for the Cyclic Scheduling of a Plant 621
Francois Warichet and Yves Pochet

Assigning Part-time Teachers to Courses via a Stable Marriage Algorithm 625
Vineet Bist, Hoang Do, Navin Sharma and George M. White

Author Index 628

Part I.

Invited Speakers

The Job-Shop Problem: Old and New Challenges

Peter Brucker
Universität Osnabrück, Albrechtstr. 28a, 49069 Osnabrück, Germany,
pbrucker@uni-osnabrueck.de

The job-shop problem is one of the most difficult classical scheduling problems. An instance with ten jobs to be processed on ten machines formulated in 1963 was open for more than 25 years. It was finally solved by a branch-and-bound algorithm. Very simple special cases of the job-shop problem are already strongly NP-hard.

After a short review of these old challenges we consider practical applications like problems in flexible manufacturing, multiprocessor task scheduling, robotic cell scheduling, railway scheduling, air traffic control which all have an underlying job-shop structure. Methods to solve these problems and new challenges in connection with them are indicated.

Keywords: Job-shop scheduling problem, complexity, branch-and-bound algorithm, local search, flexible manufacturing, multi-processor task scheduling, robotic cell, railway scheduling, air traffic control.

1 Introduction

The job-shop problem can be formulated as follows. Given are m machines $M_1, M_2, ..., M_m$ and n jobs $J_1, J_2, ..., J_n$. Job J_j consists of n_j operations $O_{ij} (i = 1, ..., n_j)$ which have to be processed in the order $O_{1j}, O_{2j}, ..., O_{n_j j}$. It is convenient to enumerate all operations of all jobs by $k = 1, ..., N$ where $N = \sum_{j=1}^{n} n_j$. For each operation $k = 1, ..., N$ we have a processing time $p_k > 0$ and a dedicated machine $M(k)$. k must be processed for p_k time units without preemptions on $M(k)$. Additionally a dummy starting operation 0 and a dummy finishing operation $N + 1$, each with zero processing time, are introduced. We assume that for two succeeding operations $k = O_{ij}$ and $s(k) = O_{i+1,j}$ of the same job $M(k) \neq M(s(k))$ holds. Let S_k be the starting time of operation k. Then $C_k = S_k + p_k$ is the finishing time of k and (S_k) defines a schedule. A schedule (S_k) is feasible if for any succeeding operations k and $s(k)$ of the same job $S_k + p_k \leq S_{s(k)}$ holds and for two operations k and h with $M(k) = M(h)$either $S_k + p_k \leq S_h$ or $S_h + p_h \leq S_k$. One has to find a feasible schedule (S_k) which minimizes the makespan $\max_{k=1}^{N} C_k$.

The flow-shop scheduling problem is a special case of the job-shop scheduling problem in which each job has m operations and the $i - th$ operation of each job has to be processed on machine M_i.

The paper is organized into four sections. Section 2 describes how to present solutions for the job shop scheduling problem and discusses some old challenges. New challenges arise in connection with applications. Section 3 describes some of these applications and shows how the classical job-shop scheduling problem needs to be extended in connection with these applications. Also corresponding solution methods are indicated. The last section contains some conclusion.

2 The job-shop problem: Old challenges

2.1 Solution representation

To solve the job-shop scheduling problem on each machine M_l one has to sequence all operations to be processed on M_l. Given all these machine sequences a corresponding left shifted schedule (S_k) (if it exists) can be constructed as follows: Let (V, A, d) be the directed network where:

1. V is the set of all operations $0, 1, ..., N, N+1$.
2. $(i, j) \epsilon A$ if and only if one of the following conditions is satisfied:
   - i and j belong to the same job and j is an immediate successor of i,
   - $i = 0$ and j is the first operation of a job or i is the last operation of a job and $j = N+1$,
   - i and j are processed on the same machine and j is sequenced immediately after i.

   The first two types of arcs are called conjunctive arcs, the last type of arcs are called fixed disjunctions.
3. $d_{ij} = p_i$ for all $(i, j) \epsilon A$.

The machine sequences define a feasible schedule if and only if (V, A, d) contains no positive cycle. In this case the lengths S_k of the longest $0-k$ paths define a feasible schedule (S_k). A longest path from 0 to $N+1$ is called critical. The length of a critical path is equal to the makespan of the schedule (S_k).

2.2 Complexity

The job-shop problem is one of the hardest scheduling problems. Only a few special cases are polynomially solvable, e.g.

- the flow-shop scheduling problem with two machines,
- the job-shop scheduling problems with two jobs,
- the job-shop scheduling problem with two machines and $p_k = 1$ for all operations k

Slightly generalized versions of these problems like

- the flow-shop scheduling problem with three machines,
- the job-shop scheduling problem with three machines and three jobs,
- the job-shop scheduling problem with three machines and $p_k = 1$ for all operations k

are already strongly NP-hard.

More detailed lists containing the hardest job-shop scheduling problems which are polynomially solvable and the easiest problems which are already NP-hard together with corresponding references can be found under http://www.mathematik.uni-osnabrueck.de/research/OR/class.

2.3 Old challenges

Besides questions concerning the complexity of special cases of the job-shop scheduling problem one of the old challenges was to find an optimal solution of a benchmark problem with 10 jobs and 10 machines. This benchmark problem was introduced in the book of Muth and Thompson [13] and could be solved only 25 years later by Carlier and Pinson [7]. Carlier and Pinson developed a clever branch-and-bound algorithm which uses constraint propagation techniques. Since then other enumerative methods have been developed, cf. the surveys of Pinson [15], Blazewicz et al. [2], and Jain & Meran [8]. Later Brinkkötter & Brucker [3] have solved 15×15 instances. It seems to be hopeless to solve larger instances by exact methods. One has to use heuristics. The best known heuristics for the job-shop scheduling problem are tabu search heuristics (cf. Nowicki & Smutnicki [14]).

3 Applications and new challenges

The job-shop problem is a basic scheduling model. Real world applications usually lead to more complex situations. We will introduce some of these applications and describe the extensions needed to cover them. Methods to solve these problems and corresponding new challenges are indicated.

3.1 Applications

Next we describe some applications which lead to extended job-shop scheduling models.

3.1.1 Flexible manufacturing

In a flexible manufacturing system the machines are equipped with tools. An operation can be processed on machines only which are equipped with the tools needed to process the operation. In other words, associated with each operation i is a set $\mu(i)$ of machines which can process i. One has to assign to each operation a machine in $\mu(i)$ and to solve the resulting job-shop problem.

3.1.2 Multiprocessor task scheduling

Tasks (operations) are instructions of a computer program (job). We assume that the tasks are to be performed one after the other. Associated with each task i is a set $\mu(i)$ of processors which are simultaneously needed to perform the task, i.e. two tasks i and j cannot be processed at the same time if $\mu(i) \cap \mu(j)$ is non-empty. If we want to schedule the instructions of a loop with a large number of repetitions we have to solve a cyclic scheduling problem.

3.1.3 Robotic cell scheduling

A robotic cell is a flow-shop or job-shop scheduling problem in which the jobs must be transported from machine to machine. Transports are performed by one or more robots. We have to assign the transport operations to the robots and to schedule both the machine and robot operations. Notice that empty moves of a robot may be necessary, e.g. if a robot moves a job from machine A to B and then another job from machine C to D it has to move empty from machine B to C before the second transport operation can be performed. Usually there is no buffer or buffers with only limited capacity to store the jobs outside the machines (or the robots).

If there are only few jobs to be processed in large numbers then periodic schedules are of interest. This leads to cyclic scheduling problems.

3.1.4 Railway scheduling

A railway scheduling problem can be modeled by a job-shop scheduling problem by dividing the railway network into track segments. A job corresponds with a train going from some origin to a destination. The operations of a job correspond with the track segments on the route of the train. The processing time of an operation is the time needed to pass the track segment. Only one train can enter a track segment. Thus, the next track segment on the route of the train must be empty before the train can enter. Otherwise the train must wait blocking the current track segment. Stopping or no-stopping has an influence on the processing time. On the other hand processing times have an influence on stopping. To resolve this problem simulation is used.

3.1.5 Air traffic control

The air space is divided into Air Traffic Control Sectors (ATCS). Each ATCS has a given capacity. Furthermore, there are n planes $j = 1, ..., n$. For each plane we have a route given by a sequence of ATCS´s with flight times for passing the ATCS´s. There is also given a time window $[a_j, b_j]$ in which a plane j must start and a best starting time t_j in $[a_j, b_j]$. For each starting time $t \in [a_j, b_j]$ the value $f_j(t)$ defines the cost for starting at time t. f_j is a function which increases with the distance between t_j and t. The planes correspond to jobs and the sequence of ATCS´s correspond to the operations of the job. However no-wait constraints must be satisfied, i.e when an operation finishes the next operation must start immediately. Also the makespan objective function has to be replaced by the function $\sum_{j=1}^{n} f_j(S_j)$ where $S_j \in [a_j, b_j]$ is the departure time of plane j. One has to fix the departure times of all planes such that the ATCS capacities are not violated and the total costs are minimized.

3.2 Extensions of the model

In the previous section we have introduced some applications which lead to extensions of the job-shop problem. In this section we will describe such extensions.

3.2.1 Assignment and scheduling

To solve complex job-shop scheduling problems we often have to assign (transportation) operations to machines (robots) and to solve the resulting scheduling problem by sequencing all operations to be processed on the same machine (robot). This is done in two stages or simultaneously (cf. [11]).

3.2.2 Positive and negative time-lags

The relation $S_i + p_i \leq S_j$ may be replaced by $S_i + d_{ij} \leq S_j$. Depending on the sign of d_{ij} this relation has different meanings: if $d_{ij} > 0$ then d_{ij} is a minimal distance between S_i and S_j (positive time-lag). If $d_{ij} < 0$ then then we have $S_i \leq S_j + |d_{ij}|$ (negative time-lag). Thus, if additionally $S_i \geq S_j$ then $|d_{ij}|$ is a maximal distance between S_j and S_i. Positive or negative or zero time-lags may be used to model

- release times and deadlines,
- exact relative timing, especially no-wait constraints,
- setup times,
- machine unavailability's,

- maximum lateness objectives by makespan objectives.

3.2.3 Blocking operations

An operation i is called a blocking operation if there is no buffer to store i after finishing on $M(i)$. Thus, if the next machine on which the successor operation $s(i)$ of i must be processed is occupied by another job then i must stay on $M(i)$ until the other job leaves the next machine. During this stay $M(i)$ is blocked for other jobs. Let i and j be two operations to be processed on the same machine and assume that j is scheduled after i. If i is a blocking operation then j cannot start before the successor operation $s(i)$ of i is started. Therefore we have the constraint $S_{s(i)} \leq S_j$. If i is non-blocking then we have $S_i + p_i \leq S_j$. The last operation of a job is usually a non-blocking operation. These concepts have be discussed in [10].

3.2.4 General objective function

Let C_j be the finishing time of the last operation of job j. Instead of minimizing the makespan $\max C_j$ one may be interested in minimizing an objective function $f(C_1, ..., C_n)$. The following cases are of special interest (cf.[6]):

- f is monotone non-decreasing,
- $f(C_1, ..., C_n) = f_1(C_1) + ... + f_n(C_n)$ where the f_j are arbitrary cost functions.

3.2.5 Cyclic scheduling

In a cyclic machine scheduling problem each operation must be repeated infinitely often. By $< i, k >$ we denote the k-th occurrence of operation i. A schedule is defined by the starting times $t(i, k)$. It is called periodic with cycle time α if $t(i, k) = t(i, 0) + \alpha k$ for all operations i and integers k. There are generalized precedence constraints between pairs (i, j) of operations defined by

$$t(i, k) + d_{ij} \leq t(j, k + h_{ij}) \qquad \text{for all integers } k.$$

d_{ij} is called delay and h_{ij} is called height. All occurrences of operation i must be processed on a dedicated machine $M(i)$. For all operations i, j with $M(i) = M(j)$ and all integers k, l the machine constraints

$$t(i, k) + p_i \leq t(j, l) \text{ or } t(j, l) + p_j \leq t(i, k)$$

must be satisfied. One has to find a cyclic schedule satisfying the generalized precedence constraints and machine constraints which minimizes the cycle time.

A cyclic job-shop scheduling problem is as special cyclic machine scheduling problem. It can be defined by specializing the generalized precedence constraints in the following way:

- Each conjunctive arc (i, j) induces a constraint $t(i, k) + p_i \leq t(j, k)$, i.e. a generalized precedence constraint with delay $d_{ij} = p_i$ and height $h_{ij} = 0$.
- Additionally a return arc $(N+1, 0)$ is introduced with a corresponding generalized precedence constraint $t(N+1, k) \leq t(j, k+h_{N+1,0})$, i.e. with delay 0 and height $h_{N+1,0}$ which is considered as a parameter.

Depending on the height $h_{N+1,0}$ optimal cyclic schedules with different cycle times are provided. The optimal cycle time is a non-increasing function of $h_{N+1,0}$. The problem is equivalent to the job-shop problem with makespan objective function if $h_{N+1,0} = 1$. Increasing $h_{N+1,0}$ provides more compact cyclic schedules usually with smaller cycle times.

There are other possibilities to introduce return arcs (cf. Brucker & Kampmeyer[5]).

3.3 Solution methods

The following techniques have been applied to solve the job-shop problem and its generalizations:

- mixed integer linear programming,
- branch-and-bound,
- heuristics.

The first two methods may provide optimal solutions. They are successful if applied to small instances. In most applications we have medium or large sized instances. This implies that one has to apply heuristics. The most successful heuristics are local search heuristics, especially tabu search (cf.[14]). In the next section we will discuss such heuristics in more detail.

3.4 Local search methods

Local search is an iterative procedure which, starting with some initial solution, moves from one solution to the next until a stopping condition is satisfied. The overall goal of this search is to improve the solutions. We have

- a search space S, and
- a neighborhood structure $N : S \to 2^S$.

For each $s \in S$ the set $N(s) \subseteq S$ describes the subset of solutions which can be reached from s in an iteration.

3.4.1 Solution representation and the search space

A solution can be defined by

- an assignment of operations to machines, and
- for each machine M_i a sequence π_i of all operations to be precessed on M_i.

Given the assignment and sequences $\pi = (\pi_i)$ a corresponding optimal schedule (if it exists) can be calculated by solving

- a longest path problem for monotone objective functions $f(C_1, ..., C_n)$ (as indicated in Section 2.1), or
- a min-cut problem for objective functions of the form $f_1(C_1) + ... + f_n(C_n)$ (cf. [12]), or
- parametric longest path problems with the cycle time as parameter for a cyclic scheduling problem ([1], [9]).

3.4.2 Moves

Moves are performed

- by changing the assignment (e.g. by moving an operation from one machine to another), or
- by changing a sequence (e.g. by shifting an operation or swapping two operations).

This can be done hierarchically or simultaneously. In the following we assume that we want to minimize the makespan. Furthermore assume that the assignments and sequences have been fixed such that a corresponding feasible schedule exists. Let CP be a longest (critical) path in the corresponding network. Then it can be shown that changing the sequence on one machine by reversing an arc in CP (if such an arc exists) will provide again a feasible schedule. A neighborhood which is restricted to such moves is opt-connected, i. e. it is possible to reach an optimal solution from any feasible solution by applying a finite sequence of moves of this type. Unfortunately, many moves which change the sequence on one machine by reversing an arc in a critical path do not change the critical path length. For these reasons a more powerful neighborhood based on blocks has been invented. A block is a sequence $i_1, i_2, ..., i_k$ of at least $k > 1$ successive operations of a critical path assigned to the same machines where k is maximal. A more powerful neighborhood is provided by the following theorem.

Blocktheorem*: To improve the makespan, at least one operation of a block B must be shifted in the corresponding machine sequence before the first or the last operation in B.*

The neighborhood defined by moves described in the Blocktheorem provides good results. However, it is a long-standing open question wether such a neighborhood is opt-connected. The Blocktheorem can be generalized to job-shop problems with blocking operation [4] and to cyclic scheduling problems with or without blocking operations [9].

3.5 New challenges

There is much room to improve existing heuristics and to develop new heuristics for complex job-shop scheduling problems. For example the following areas and their combinations are of interest:

- job-shop scheduling problems with buffers of limited capacity,
- cyclic job-shop scheduling problems, and
- routing and sequencing problems.

When applying local search heuristics an important issue is to avoid deadlock when moving from one solution to the next. Repair mechanisms have been developed which can be applied when a move turns a feasible solution into a deadlock situation. However, often the quality of solutions provided by such repair mechanisms have a poor quality.

Another issue is to improve local search methods for combined assignment and sequencing problems. In [4] buffer slot assignments are replaced by sequences which allow to calculate the assignments in polynomial time. It is an interesting question wether similar ideas can be applied to other combined assignment and sequencing problems.

4 Conclusion

The classical job-shop scheduling problem is an optimization problem which has some basic structure. Practical problems are usually more complex. We have presented models for complex job-shop scheduling problems. Based on these models heuristics to solve such complex scheduling problems have been developed. Most promising heuristics are local search algorithms (especially tabu search). Structural properties of the problem should be exploited when developing these heuristics.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin (1993), *Network Flows*, Prentice Hall, Englewood Cliffs.

[2] J. Błażewicz, W. Domschke, E. Pesch (1996), The job shop scheduling problem: conventional and new solution techniques, *European Journal of Operational Research* **93**, 1 – 33.

[3] W. Brinkkötter, P. Brucker (2001), Solving open benchmark instances for the job shop problem by parallel head-tail adjustments, *Journal of Scheduling*, 53 – 64.

[4] P. Brucker, S. Heitmann, J. Hurink, T. Nieberg (2006), Job-shop scheduling with limited capacity buffers, *OR Spectrum* **25**, 151 – 176.

[5] P. Brucker, T. Kampmeyer (2007), Cyclic Scheduling Problems and their Applications, submitted to *Discrete Applied Mathematics*.

[6] P. Brucker, S. Knust (2006), *Complex Scheduling*, Springer, Heidelberg.

[7] J. Carlier, E. Pinson (1989), An algorithm for solving the job-shop problem, *Management Science* **35**, 164 – 176.

[8] A.S. Jain, S. Meeran (1999), Deterministic job-shop scheduling: past, present and future, *European Journal of Operational Research* **113**, 390 – 434.

[9] T. Kampmeyer (2006), *Cyclic scheduling problems*, PhD-Thesis, Fachbereich Mathematik/Informatik, University of Osnabrück.

[10] A. Mascis, D. Pacciarelli (2000), Job-shop scheduling with blocking and no-wait constraints, *European Journal of Operational Research* **143**, 498 – 517.

[11] M. Mastrolilli, L.M. Gambardella (2000), Effective neighborhood functions for the flexible job shop problem, *Journal of Scheduling* **3**, 3 – 20.

[12] R.H. Möhring, A.S. Schulz, F. Stork, M. Uetz (2003), Solving project scheduling problems by minimum cut computations, *Management Science* **49**, 330 – 350.

[13] J.F. Muth, G.L. Thompson (1963), Industrial Scheduling, Prentice Hall, Englewood Cliffs.

[14] E. Nowicki, C. Smutnicki (2005), An advanced tabu search algorithm for the job-shop problem, *Journal of Scheduling* **8**, 145 – 159.

[15] E. Pinson (1995), The job-shop scheduling problem: a concise survey and some recent developments, in: P. Chretienne, E. G. Coffman, J.K. Lenstra, Z. Liu (eds.), *Scheduling Theory and its Applications*, John Wiley, 277 – 293.

Walk the (Time-)Line: Scheduling and Execution

Amedeo Cesta
Italian National Research Council, Institute for Cognitive Science and Technology
Via S. Martino della Battaglia, I-00185, Rome, Italy, amedeo.cesta@istc.cnr.it

At its core, a project scheduling problem consists in synthesizing functions that represent availability of resources over time (*resource timelines*) which are consistent with both resource bounds and temporal constraints specified in the problem.

This talk begins with a review of the artificial intelligence constraint-based approach to project scheduling, also referred to as *precedence-constraint posting*. This approach is based on the ability to solve the problem by reasoning on the representation of temporal constraints and, on top of it, of resource timelines and their current contention peaks.

The talk will then introduce the complementary problem of schedule execution under uncertainty, where the same type of reasoning can be used to obtain schedules which are resilient to execution-time contingencies. Here, two types of solutions are distinguished, namely *flexible* and *partial order* schedules, which lead to timelines with different degrees of robustness with respect to unexpected variations at execution time.

A third part of this talk will be dedicated to a survey of recent developments of our work, including relevant applications of the presented techniques in the context of space applications.

Keywords: Reasoning about Time and Resources, Constraint-based Scheduling, Contention-based Heuristics, Schedule Execution Uncertainty

Scheduling with Batch Compatible Tasks

Gerd Finke
Laboratoire G-SCOP, University Joseph Fourier Grenoble, France, gerd.finke@g-scop.inpg.fr

We analyze batch-scheduling problems that arise in connection with certain industrial applications. The models concern processing on a single max-batch machine with the additional feature that the tasks of the same batch have to be compatible. Compatibility is a symmetric binary relation – the compatible pairs are described with an undirected "compatibility graph", which is often an interval graph according to some natural practical conditions that we present. We consider several models with varying batch capacities, processing times or compatibility graphs and summarize all known results.

Keywords: Batch-scheduling, Task compatibilities, Interval graphs, Bounded coloring.

1 Introduction

This presentation is mainly based on the article [7] and the thesis [12] and describes the developments of this branch since the original application [4] in 1999. A *batch machine* refers to a machine that can process several tasks simultaneously. We consider here the so-called *max-batch* or *parallel-batch* (sometimes abbreviated *p-batch*) machine, where the processing time of a group of tasks, called a *batch*, is the longest processing time of the tasks it contains. The initial motivation for this branch of scheduling theory was the scheduling of semiconductor burn-in operations [11]. Intensive research has subsequently been developed on this subject for various scheduling objectives and additional constraints, see for instance the surveys [5] and [15].

Here we focus on minimizing the makespan C_{max}. In addition, we assume that the tasks in the same batch have to be *compatible*, for instance they must share similar physical properties(form, weight, etc). The problem that we want to analyze may be formulated as follows. There are n independent tasks T_j $(j = 1, \ldots, n)$ to be scheduled on a single max-batch machine. The batch machine has *capacity* b, which means that at most b tasks can be processed simultaneously (b may be finite or infinite). Each task T_j has a (minimal) processing time p_j. A batch B has processing time $p(B) = \max\{p_j : T_j \in B\}$ and all tasks in the same batch start and finish at the same time. Preemption is not allowed. Tasks in the same batch have to be *pairwise compatible*. This relation is represented by a *compatibility graph* $G = (V, E)$ where V is the set of tasks and a pair of tasks is an element of the edge set E if and only if they are compatible. By definition, a batch forms a *clique* (a complete subgraph, not necessarily maximal) in the compatibility graph G. Since all tasks have to be executed, the problem is to find a decomposition of G into cliques $B_1, B_2, \ldots, B_l$ where l is not known in advance, such that the schedule length $C_{max} = \sum_i p(B_i)$ is minimized. These batches may then be scheduled in any order, without any idle time.

The concept of scheduling with batch compatible tasks has been treated in [1]–[4], [7, 9, 12, 13] for general graphs and also for some special graphs. This theory is related to chromatic scheduling [6] where the complementary graph (graph of incompatibilities) is considered, leading to a graph coloring problem. For chromatic scheduling, there are usually no capacity constraints.

Our motivation for this type of batch scheduling problem originates in several industrial applications. The first one comes from the sheet metal industry [4], another from a rolling-mill [14], and finally from the process of tire making [13]. For these quite different industrial applications, we

shall illustrate that interval graphs occur quite naturally as compatibility graphs in batch processing. In the following Section 2, the batch scheduling models are formulated that result from these applications.

2 Batch Scheduling Models

We consider batch scheduling problems on a single max-batch machine with task compatibilities. We use the notations:

$$1 \,/\, p\text{-}batch,\, G = \beta_1,\, \beta_2 \,/\, C_{max}$$

where the compatibility graph is specified by the parameter β_1; we use $\beta_1 = INT$ for an interval graph and $\beta_1 = (V, E)$ for a general graph. The parameter β_2 specifies the batch capacity b as follows: "$b = k$" for a fixed capacity; "$b < n$" for a variable finite capacity b which is part of the input; or β_2 is void for infinite capacity. There may be other parameters β_i representing additional restrictions, for instance structured processing times, release dates, etc. Referring to the various applications, one is lead to the following problem types.

(P1) $1 \,/\, p\text{-}batch,\, G = INT,\; p_j = 1 \,/\, C_{max}$

Problem $(P1)$ has also been solved approximately in [4], [9] for graphs that are slightly more general than interval graphs.

(P2) $1 \,/\, p\text{-}batch,\, G = INT,\, b < n,\, p_j = 1 \,/\, C_{max}$

We get two more models if we allow arbitrary processing times:

(P3) $1 \,/\, p\text{-}batch,\, G = INT \,/\, C_{max}$

(P4) $1 \,/\, p\text{-}batch,\, G = INT,\, b < n \,/\, C_{max}$

The aim of this presentation is to give an overview of the solution approaches to these problems and their extensions to more general graphs.

References

[1] M. Boudhar (2003), Scheduling a batch processing machine with bipartite compatibility graphs, *Mathematical Methods of Operations Research* **57**, 513 – 327.

[2] M. Boudhar (2003), Dynamic Scheduling on a Single Batch Processing Machine with Split-Compatibility Graphs, *J. of Mathematical Modelling and Algorithms* **2**, 17 – 35.

[3] M. Boudhar, G. Finke (2000), Scheduling on a batch machine with job compatibilities, *Belgian J. of Operations Research, Statistics and Computer Science* **40**, 69 – 80.

[4] N. Brauner, C. Dhaenens-Flipo, M.-L. Espinouse, G. Finke,H. Gavranovic (1999), Decomposition into parallel work phases with application to the sheet metal industry, *Proc. International Conf. on Industrial Engineering and Production Management (IEPM'99)* **1**, Glasgow, 389 – 396.

[5] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C. Potts,T. Tautenhahn, S. van de Velde (1998), Scheduling a batching machine, *J. of Scheduling* **1**, 31 – 54.

[6] M. Demange, D. de Werra, J. Monnot, V.T. Paschos (2001), Time slot scheduling of compatible jobs, *Cahier du Lamsade* **182**, Université Paris IX - Dauphine, Paris, OR Working Paper, EPF Lausanne.

[7] G. Finke, V. Jost, M. Queyranne, A. Sebő (2007), Batch processing with interval graph compatibilities between tasks, to appear in *DAM*.

[8] F. Gardi (2005), Ordonnancement avec exclusion mutuelle par un graphe d'intervalle ou une classe apparentée : complexité et algorithmes, *Ph.D. Thesis*, *Laboratoire d'informatique fondamentale - Faculté des sciences de Luminy, Marseille*.

[9] H. Gavranovic, G. Finke (2000), Graph partitioning and set covering for the optimal design of a production system in the metal industry, *Proc. Second Conference on Management and Control of Production and Logistics (MCPL'2000)* **2**, Grenoble, 603 – 608.

[10] P. Hansen, A. Hertz, J. Kuplinsky (1993), Bounded vertex colorings of graphs, *Discrete Mathematics* **111**, 305 – 312.

[11] D.S. Hochbaum, D. Landy (1997), Scheduling semiconductor burn-in operations to minimize total flowtime, *Operations Research* **45**, 874 – 885.

[12] V. Jost (2006), Ordonnancement chromatique: Polyèdres, Complexité et Classification, Ph.D. Thesis, *Université Joseph Fourier, Grenoble*.

[13] A. Kamgaing Kuiteing, A. Oulamara, G. Finke (2006), Flowshop scheduling with a batching machine and task compatibilities, *Proc. Conf. INCOM, St. Etienne*.

[14] P.L. Nguyen (1997), Planification tactique de la production : approche hiérarchisée pour une classe d'entreprises de sous-traitance et application au cas de laminage à froid, Ph.D. Thesis, *Université Joseph Fourier, Grenoble*.

[15] C.N. Potts, M.Y. Kovalyov (2000), Scheduling with batching: A review, *European Journal of Operational Research* **120**, 228 – 240.

Scheduling in Highly Uncertain Environments

Rhonda Righter
IEOR Department, University of California, Berkeley, CA USA RRighter@IEOR.Berkeley.edu

We consider scheduling problems that arise on the Internet, where uncertainty and variability are high, and ask the following questions. How do we deal with extreme uncertainty? How might variability increase the range of scheduling options? When does more variability help us? When is it worth it to do extra monitoring to collect more information (and when not)? How can we compensate for a lack of information? We consider these questions for two particular Internet applications: router scheduling and computational grid scheduling.

Keywords: multi-processor scheduling, grid scheduling, stochastic scheduling

1 Introduction

In many manufacturing and service environments, there is little variability, and indeed, deterministic approximations often perform well in scheduling algorithms. In other situations randomness must be explicitly taken into account, but the variability can be captured in fairly simple models with well-behaved distributions. However, the Internet exhibits a whole new level of uncertainty in many dimensions. Internet traffic is highly variable in terms of heavy-tailed distributions, self-similar traffic, and long-range dependencies. Moreover, when many heterogeneous resources, each under the control of their own local users, are combined over the Internet into a grid to solve large-scale scientific and business problems, there are many sources of dynamic variability in terms of response times, resource availabilities, and computation errors. We consider the opportunities and challenges involved with scheduling in two Internet applications with high uncertainty. In terms of the standard scheduling problem of assigning and sequencing jobs on processors, for the first application, scheduling files on an Internet router, it is the jobs (files) that exhibit most of the variability, whereas for the second, grid scheduling, the variability and uncertainty is primarily due to the processors.

We first examine Internet router scheduling, in which there is a mixture of files of highly variable length that must be transferred, and file sizes are not known in advance. Indeed, the distribution of file sizes being sent over the Internet often has infinite variance [1]. If file sizes (job processing times) are variable enough, in a sense to be defined precisely later, then the usual nonpreemptive schedules such as first-come first-served can be the worst performing schedules. Fortunately, for Internet routers, and in contrast to most manufacturing and traditional service systems, preemption is a viable scheduling option, and can be done at little or no cost. Using preemption intelligently can greatly improve performance when processing time variability is high. In addition, for some scheduling disciplines that compensate for file size variability, performance may increase with increasing variability.

We also consider grid computing, in which a network of computers is integrated to create a very fast virtual computer. This computing paradigm is becoming ever more prevalent because of its ability to harvest a large amount of computing power at relatively low cost [3]. Grid computing creates a fast virtual computer from a network of computers by using their idle cycles. Although grid computing is a successor to distributed computing, the computing environments are fundamentally different. For distributed computing, resources are homogeneous and are reserved,

leading to guaranteed processing capacity. On the other hand, grid environments are highly unpredictable. The computers are heterogeneous, their capacities are typically unknown and changing over time, they may connect and disconnect from the grid at any time, and they may produce incorrect outputs [2].

The basic grid scheduling problem looks like a standard multi-processor scheduling problem, and if we had accurate estimates of response times for the different processors, the problem would reduce to a bin packing or load balancing problem. However, such response-time estimates are difficult to obtain and are inherently unreliable because of the dynamics of the system. On the other hand, the grid computing environment gives us a powerful new option for scheduling that does not exist (nor would it make sense) in manufacturing and standard service applications. This is the option of replicating jobs, i.e., sending the exact same work to two or more different processors that operate independently, and taking the results from whichever finishes first. When processing times are variable enough (again in a sense to be made precise later), replication makes sense in terms of minimizing mean flow times and/or makespans. In addition, it has other nice properties in terms of ease of implementation, low information requirements, robustness and adaptability, fairness, predictability, and synchronization.

Though we use the terms variability and uncertainty somewhat interchangeably to avoid repeating both words, the concepts are distinct. Roughly, variability is the range in possible values random variables can take (as measured by, e.g., range, variance, covariance), and while the random variables themselves may be highly variable, the measures of variability are generally precise. On the other hand, we may not know the parameters of the distribution (this is sometimes referred to as ambiguity), or even the possible distributional families of the random variables. Not knowing these distributions, or even the number of random variables involved, may be called uncertainty.

2 Internet Router Scheduling

Files transmitted over the Internet, or pages requested for download, exhibit extreme variability in terms of their file sizes (processing times), and often have infinite variance [1]. File types range from E-mail to video to software downloads, and file-size variability is large even within a file type. In addition, file sizes are often unknown. When file sizes are known, SRPT (shortest remaining processing time first) is the optimal scheduling policy for file transmissions for a single Internet router (a single server). Such a policy minimizes the number of jobs in the system path-wise [10], and hence minimizes mean response time. On the other hand, when file sizes are not known and are highly variable, a naive strategy such as FCFS (first come first served) will perform badly, and will produce infinite mean response time when the file size variance is infinite, even at low loads..

A reasonable model for the file-size distribution is a DFR (decreasing failure rate) distribution. For a random variable X, that we suppose continuous for simplicity, with distribution $F(x)$, tail distribution $\bar{F}(x) = 1 - F(x)$, and density $f(x) = F'(x)$, its failure (or hazard) rate at time (or age) t is $h(t) = f(t)/\bar{F}(t)$ (e.g., [6] or [11]). Hence, X is DFR if $h(t)$ is decreasing in t. Roughly, if a job has a DFR processing time, it is less and less likely to complete soon as you work on it. For example, as the time to complete a file transfer increases, it is more likely to be a long file (e.g. video rather than E-mail), so it is less likely to complete soon. DFR distributions naturally arise when mixing distributions, and imply a coefficient of variation of at least 1. Examples of DFR distributions are Pareto, hyperexponential, lognormal, and Weibull and gamma with $\alpha \leq 1$. If file sizes have a DFR distribution, then LAS (least attained service time first) stochastically maximizes the number of files completed by any time (and hence minimizes mean flow time), and any nonpreemptive discipline, such as FCFS, stochastically minimizes the number of files completed by any time [9]. (The reverse holds for IFR, i.e., increasing failure rate, distributions.) The LAS

discipline, also known as Foreground-Background, always works on the jobs with the smallest ages (the youngest jobs), in a processor sharing fashion if there are ties. So, for example, new arrivals preempt jobs currently being processed. When an interrupted job is resumed, it is resumed where it left off (so its remaining processing time has the same distribution as if it had not been preempted). For DFR processing times, LAS corresponds to SERPT (shortest expected remaining processing time first), so it is a means of sorting jobs by expected remaining size based on their age. The optimality of LAS for DFR processing time distributions holds for arbitrary arrival processes, i.e., regardless of the levels of uncertainty, variability, and long-range dependence in arrival times.

The LAS discipline can be difficult to implement, and will be impossible if there is no mechanism for keeping track of the ages of jobs. An alternative discipline that has some potential to sort jobs by size is the LCFS-pr (last come first served preempt resume). In this discipline the most recent arrival is always served, preemptively, and interrupted jobs are resumed where they left off. Under LCFS-pr a short job has the potential to finish before the next arrival. Under LCFS-pr no information on job ages is required; we need only maintain a queue. It seems reasonable to expect that LCFS-pr will perform well, and perhaps optimally in an appropriate sense, when processing times are NWU (new worse than used). A random variable X is NWU if

$$P\{X > x\} \leq P\{X - t > x | X > t\} \; \forall t,$$

and this is weaker than DFR (and also implies a coefficient of variation of at least 1). Note that "worse" is in a reliability sense; if a job's processing time is NWU, then the remaining processing time (life time in reliability terms) of a job that has received no processing is stochastically shorter than that of a job that has received some processing already.

A relevant performance measure for an Internet router is the maximal queue length during a busy period, M. Since buffer sizes are typically finite, this measure is critical in helping us design a good buffer size. For this performance measure, more variability in processing times improves performance under LCFS-pr [5], regardless of whether processing times are DFR or NWU. Here we measure variability in the convex ordering sense. In particular, we say that for two random variables X and X', $X' \geq_{cx} X$ if and only if $Ef(X') \geq Ef(X)$ for all convex functions f. Note that

$$X' \geq_{cx} X \Rightarrow EX' = EX \text{ and } Var(X') \geq Var(X).$$

Then, for two $M^X/G/1$ LCFS-pr queues (with batch Poisson arrivals) having generic service times S and S', and corresponding busy-period maximal queue lengths M and M',

$$S' \geq_{cx} S \Rightarrow M' \leq_{st} M \Leftrightarrow P\{M' > b\} \leq P\{M > b\} \; \forall b.$$

This is somewhat surprising because generally we expect variability to degrade performance in queues, and indeed, this is the case for non-preemptive disciplines such as FCFS. For example, Miyazawa (1990) and Miyazawa and Shanthikumar (1991) show that for the finite-buffer $M^X/G/1/b$ queue under non-preemptive disciplines, the loss rate will be larger when service times are more variable in the convex sense.

3 Computational Grid Scheduling

Grid computing creates a fast virtual computer from a network of computers by using their idle cycles. Examples include networks such as the TeraGrid, a transcontinental supercomputer set up at universities and government laboratories and supported by NSF, ad hoc networks within universities or laboratories, and applications on the Internet that take advantage of the unused computing

capacity of idle desktop machines such as SETI@home, Folding@home, and Einstein@home. Grids are characterized by a high degree of uncertainty in processing times, machine availabilities, and output correctness, and these things are heterogeneous across computers and time.

The "processing time" of a job on a grid computer is the time from sending the job (or the input data) until receiving the output data. From an individual computer's point of view, the grid job is typically a low priority job that will only be done when the computer would otherwise be idle, and only when the computer is on or connected to the network. Therefore, the processing time may include queueing times and down times, and the job could even be deleted from the computer's queue before completion, for example if the computer is powered off or crashes. Because of firewalls and privacy concerns, the grid scheduler will typically have very little information about the status of an individual computer or the job assigned to it. Also, the computers in a grid are highly heterogeneous (and even include Sony PlayStations in the case of Folding@home). Finally, in some cases output data may be corrupted.

As a starting assumption, we suppose that the variability in the processing times comes from the computers and not the jobs. This will be the case when jobs are performing the same subroutine on different data, as in Single-Program-Multiple-Data (SPMD) parallel programs. SPMD programs are used in many applications, including computational fluid dynamics, environmental and economic modeling, image processing, simulation, and dynamic programming. Thus, the jobs are identical. We also assume that the processing time of the same job on different computers is independent, which is reasonable in a grid environment where different computers are at different locations and belong to different entities. We further assume that, given an appropriate environmental state, future processing times are independent of past processing times.

We ignore communication delays, which is reasonable when processing times are significantly longer than communication times, and is generally the case for applications in which grid computing makes sense. For example, with SPMD applications, only the input data needs to be transmitted.

With grid computing we have the option of job replication, i.e., transmitting the same input data to multiple computers, and taking the output from the first to be completed. We assume that if a job is replicated and finishes on one computer, there is no cost for deleting that job from other computers. This assumption again is reasonable in grid environments in which the reason one computer takes longer than another to process the same job is because it is either unavailable or busy with its own, higher priority, work. We only permit preemption or deletion of a job on a computer when that job completes on some other computer. When a job is deleted from a computer, a new job can be sent to that computer.

Suppose that, given the environmental state, processing times are NWU. That is, when a job is first assigned to a computer it has stochastically shorter processing (response) times than one that was assigned earlier. Again this is not an unreasonable assumption in a grid environment where, for example, a computer that hasn't returned a result in a while may be turned off. For NWU processing times, maximal replication is optimal in terms of stochastically maximizing the departure process (the number of departures by any time t, and jointly across t) [4]. That is, we should essentially do each job sequentially, replicating each job on all available computers until a result from any computer is returned, and then replicating the next job. This will minimize mean flowtime and makespan.

Maximal replication has many nice properties that allow us to extend its optimality, given NWU processing times, to much more general models. For example, jobs may have precedences. Since without precedences sequential maximal replication in any order is optimal, with precedences, sequential replication in any order satisfying the precedences will still be optimal. Alternatively, jobs may be heterogeneous in terms of processing times or costs or deadlines or probabilities of producing correct results. All the single-machine scheduling results along with maximal replication

will hold, e.g., the optimality of SPT, weighted SPT, EDD (earliest due date first), or priority to jobs that are more likely to be done correctly, under appropriate conditions.

Maximal replication is extremely simple to implement, and is robust and adaptive; it balances the load without requiring any information about the computers. That is, it is optimal with or without that information. It easily adapts to the arrival of new computers to the grid, and it automatically fully utilizes the computers, even when the number of jobs is less than the number of computers. It is fair in the sense that no particular job will get "stuck" with a computer that is down, and it improves response-time predictability. This is important, for example, for setting QoS (quality of service) guarantees. Maximal replication also extends easily to permit error correction. For example, after replicating a single job on all computers we could wait for the first two computers to complete, and if their results agree, we would consider the job complete and replicate the next job. If they don't agree, we could continue to wait for further results from other computers.

When using a maximal replication policy, it is easy to show that performance improves if processing times become more variable in the convex ordering sense. (More variable processing times lead to a smaller minimum processing time across the computers.)

4 Conclusion

We have considered two Internet scheduling problems, router and grid scheduling, that have a high degree of uncertainty. For each application there is a scheduling option (preemption and replication respectively) that is generally not available in standard manufacturing and non-IT service systems, and that helps us deal with the variability and compensate for a lack of information. Also, for schedules that address variability, increasing the variability can improve performance. For the grid application we have seen that for schedules that are appropriate in the context of high variability, there may be no advantage to collecting more information.

References

[1] Crovella, M., Bestavros, A. (1996), Self-similarity in World Wide Web traffic: Evidence and possible causes, *Proceedings of the ACM Sigmetrics '96 Conference*, 160 – 169.

[2] M. Dobber, R. van der Mei, and G. Koole (2006), Statistical properties of task running times in a global-scale grid environment, *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid* (CCGrid 2006), 150 – 153.

[3] L. Foster and C. Kesselman, editors (1999), *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco.

[4] G. Koole and R. Righter (2007), Resource Allocation in Grid Computing, *Journal of Scheduling*, to appear.

[5] G. Koole, M. Nuyens, and R. Righter (2005), The Effect of Service Time Variability on Maximum Queue Lengths in Batch M/G/1 Queues, *Journal of Applied Probability* **42**, 883 – 891.

[6] C.-D. Lai and M. Xie (2006), *Stochastic Ageing and Dependence for Reliability*, Springer, New York.

[7] M. Miyazawa (1990), Complementary generating functions for $M^X/GI/1/k$ and $GI/M^Y/1/k$ queues and their applications to the comparison of loss probabilities, *Journal of Applied Probability* **27**, 684 – 692.

[8] M. Miyazawa and J.G. Shanthikumar (1991), Monotonicity of the loss probabilities of single server finite queues with respect to convex order of arrival or service processes, *Probability in the Engineering and Informational Sciences* **5**, 43 – 52.

[9] R. Righter and J.G. Shanthikumar (1989), Scheduling multiclass single server queueing systems to stochastically maximize the number of successful departures, *Probability in the Engineering and Informational Sciences* **3**, 323 – 333.

[10] L.E. Schrage (1968), A proof of the optimality of the shortest remaining processing time discipline. *Operations Research* **16**, 687 – 690.

[11] M. Shaked and J. G. Shanthikumar (2007), *Stochastic Orders*, Academic Press, New York.

Interval Scheduling

Frits C.R. Spieksma
Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium, frits.spieksma@econ.kuleuven.be

This abstract is based on a survey by Kolen et al. [28].

Keywords: Applications, Complexity of Scheduling Problems

1 Introduction

Scheduling problems are formulated in terms of machines and jobs. The machines represent resources and the jobs represent tasks that have to be carried out using these resources. In a traditional scheduling problem there is freedom in determining the starting times of the jobs. The scheduler uses this freedom by attempting to construct a schedule that satisfies certain constraints or even one that optimizes a certain criterion. Performance measures for this type of scheduling problem reflect how well the resources are used. For instance, minimizing the makespan measures the time it takes the machines to process all given jobs. Thus, the quality of a schedule measures the performance of a set of machines processing all given jobs. Research on these classical scheduling problems is abundantly present in the operations research literature.

This presentation deals with *interval scheduling* problems, also known as *fixed job scheduling* or *k-track assignment* problems. In this type of scheduling problem there is no freedom in determining the starting time of the job; instead this is input for the problem. Decisions that the scheduler now faces involve whether or not to accept the job and what resource to assign to it. Performance measures here can focus on the individual jobs; for instance, one may wish to maximize the total weight of the accepted jobs. Thus, performance measures in interval scheduling allow taking into account the cost of rejecting (or the profit of accepting) an individual job. Although there is a sizable amount of literature on interval scheduling problems, research on these problems has often been tailored to the application at hand, and results are scattered through literature. For instance, the well known crew scheduling problem can be seen as an interval scheduling problem (see Section 4).

Although the relevance of interval scheduling problems stems to a large extent from the variety of applications that have an interval scheduling decision as a core, recent trends in operations management also motivate their study. Elaborating on this, recent literature in operations management (see e.g. Graves et al. [22]) describes very clearly how a transition took place in the last decennia from *resource oriented* logistics (where the availability of resources dictated the planning and completion of jobs) to *demand oriented* logistics (where the jobs and their completion are more or less fixed and the appropriate resources must be found). This transition has mainly been caused by increased competition and the continuous drive to improve service to clients. In an environment where clients are becoming more important they will have more influence on the delivery date, and hence on the production date of their product. Thus, dates for production tend to become more exogeneously determined, as opposed to a more traditional setting where the plant itself could decide when to manufacture a job. This phenomenon is enhanced by current logistic developments in which production is organized by supply chains. To summarize: when phrasing the transition sketched above in scheduling terminology, it is akin to going from traditional scheduling to interval scheduling. Therefore, interval scheduling can provide a useful way to model some of the operational planning problems in current logistics.

The history of interval scheduling can be traced back to the 1950's when Dantzig and Fulkerson [14] described a tanker scheduling problem. Ford and Fulkerson [20] solve a basic interval scheduling problem (see Section 2), using Dilworth's theorem.

2 The basic interval scheduling problem

The basic interval scheduling problem is stated as follows. Given are n intervals of the form $[s_j, f_j)$ with $s_j < f_j$, for $j = 1, \ldots, n$. These intervals are the *jobs* that require uninterrupted processing during that interval. It is assumed (without loss of generality) that the s_j's and the f_j's are nonnegative integers. Two intervals (or jobs) are said to *overlap* if their intersection is non-empty, otherwise they are called *disjoint*. Further, there are machines. In the basic interval scheduling problem each machine can process at most one job at a time and is always available, i.e., each machine is continuously available in $[0, \infty)$. We assume that, when processed, each job is assigned to a single machine, thus, interrupting a job and resuming it on another machine is not allowed, unless explicitly stated otherwise. The basic interval scheduling problem is now to process all jobs using a minimum number of machines. In other words, find an assignment of jobs to machines such that no two jobs assigned to the same machine overlap while using a minimum number of machines. An assignment of (a subset of) the jobs to the machines is called a *schedule*.

It is well known that interval scheduling problems and interval graphs are related. Indeed, the graph that arises when there is a node for each interval and when there is an edge between two nodes if and only if the corresponding intervals overlap, is a so-called interval graph. Hence, the basic interval scheduling problem is in fact nothing else but finding a coloring of an interval graph; the chromatic number of an interval graph corresponds to the minimum number of machines used in the interval scheduling context.

Another observation concerning the basic interval scheduling problem is as follows. Consider a subset of jobs that pairwise overlap. Obviously, the size of this set is a lower bound for the number of machines needed to process all jobs. Given an instance of the basic interval scheduling problem, let us call the maximum size of such a set the *job overlap* of the instance. Obviously, the job overlap is a lower bound for the number of machines needed. However, using Dilworth's chain decomposition theorem, one can show that this number of machines actually suffices to process all jobs. An algorithm to construct an optimal schedule (that is, one with a minimum number of machines) is described by Ford and Fulkerson [20], who specified the so-called *staircase rule*, which is based on Dilworth's theorem. The rule involves $O(n^2)$ operations. Gupta et al. [23] propose another procedure that runs in $O(n \log n)$ time, which they show to be best possible.

3 Variants of the basic interval scheduling problem

This presentation considers different (optimization) variants of the basic interval scheduling problem. In particular, known results for the following settings are discussed:

- Interval scheduling problems where each given job needs to be carried out. In such problems, the objective is to find a minimum-cost schedule in which all jobs are scheduled. These problems are referred to as interval scheduling with required jobs (see Subsection 3.1).

- Interval scheduling problems where the number of given machines is fixed. In such problems, the jobs are given, the number of machines is fixed, and a profit is given when job j is carried out by machine i; the resulting problem is to find a maximum-profit schedule (in which not

all jobs need to be assigned). These problems are referred to as interval scheduling with given machines (see Subsection 3.2).

- Discrete interval scheduling problems (see Subsection 3.3). In this setting, instead of a single start time s_j for each job j, a discrete set of possible starting points $S_j = \{s_{j1}, s_{j2}, \ldots, s_{jk}\}$ is given for each job j.
- Online interval scheduling problems and interval scheduling problems with preemption (see Subsection 3.4).

3.1 Interval scheduling with required jobs

In this section the variant is considered where the jobs are given, and the total costs induced by carrying out all jobs on the nonidentical machines need to be minimized. (Notice that if the machines were identical, an instance of the basic interval scheduling problem arises.) There are different ways in which the machines can be nonidentical. For instance, machines may differ with respect to their *availability*. Indeed, consider the variant where to each machine i, $1 \leq i \leq m$, an availability interval $[a_i, b_i)$ is associated, where there is a given cost for using machine i, and where the goal is to minimize total costs while scheduling all jobs. This problem is denoted by *interval scheduling with machine availabilities*. The following statement applies to this variant: deciding whether a feasible schedule exists is NP-complete.

Bhatia et al. [6] consider a related setting where a machine *type* i corresponds to an an availability interval $[a_i, b_i)$, $1 \leq i \leq m$. Thus, for each availability interval there is an unbounded number of machines available instead of a single one, and there is a given cost k_i for using a machine of type i. Assuming that each job can go to each machine, Bhatia et al. [6] give a 3-approximation algorithm (and a 2-approximation algorithm in case one wants to minimize the number of machines used, i.e., in case $k_i = 1$ for all i).

Another way in which the machines can be nonidentical is by assuming that there is a *linear order* given for the machines, and, in addition, for each job j a 'maximal' machine $m(j)$, $1 \leq m(j) \leq m$, is given on which it can be processed. Thus machines $1, \ldots, m(j)$ are capable of processing job j while machines $m(j)+1, \ldots, m$ cannot process job j, $j = 1, \ldots, m$. Given a cost k_i for using machine i, the goal is to minimize costs while scheduling all jobs (recall that each machine is continuously available). The resulting problem is called the *hierarchical interval scheduling problem*. Again, deciding whether a feasible schedule exists is NP-complete. However, in case there are *types* of machines, and there is a linear order for these machine types, Bhatia et al. [6] give a 2-approximation algorithm. When there are two machine-types, Dondeti and Emmons [15] and Huang and Lloyd [25] show that (a generalization of) the resulting problem is solvable in polynomial time. The problem becomes NP-complete for three machine types (Kolen et al. [28]).

3.2 Interval scheduling with given machines

An important optimization variant is the case where one treats the machines as given and the objective is to maximize the number of (weighted) jobs that can be feasibly scheduled. This variant can be solved using a min-cost flow formulation (see Arkin and Silverberg [2] and Bouzina and Emmons [7]). In case each job has unit weight, a greedy algorithm finds a maximum number of jobs (see Faigle and Nawijn [17], and Carlisle and Lloyd [10]). If each job can only be carried out by an arbitrary given subset of the machines, the problem becomes NP-hard ([2]). Heuristics and exact algorithms are proposed by Kroon et al. [29]. In case an availability interval is associated to each machine, Brucker and Nordmann [9] describe an $O(n^{m-1})$ algorithm that maximizes the number of

jobs scheduled. Bhatia et al. [6] describe a randomized approximation algorithm achieving a ratio of $1 - \frac{1}{e}$ when there is a given profit w_j for each job j scheduled.

3.3 Discrete interval scheduling problems

Discrete interval scheduling problems constitute an interesting generalization of interval scheduling problems. Instead of a single starting time s_j for each job j, a discrete set of possible starting points $S_j = \{s_{j1}, s_{j2}, \ldots, s_{jk}\}$ is given. Of course, at most one starting time from each set S_j can be chosen.

Discrete interval scheduling problems are related to so-called *time-constrained* scheduling problems. In a time-constrained scheduling problem, a release date r_j, a deadline d_j, and a processing time p_j is given for each job j. Obviously, if $d_j = r_j + p_j$ for each job j, an interval scheduling problem arises, but also, when assuming that there is an interval for each possible realization of job j, a time-constrained scheduling problem can be modeled as a discrete interval scheduling problem. Of course, one faces here the difficulty of having to deal with, possibly, a continuum of intervals reflecting all possible starting times of a job (notice that the word 'job' refers here to a set of intervals).

Results for discrete interval scheduling problems are given by Nakajima and Hakimi [31], Nakajima et al. [32] (for a setting with *fast* and *slow* machines), Keil [26], and Spieksma [33]. More in particular, in [33], a single machine is considered, and the goal is to maximize the number of jobs selected. It is shown that this problem is APX-hard (even if $|S_j| = 2$ for each job j), that the integrality gap of a straightforward integer programming formulation equals 2, and that a greedy algorithm yields a 2-approximation. Chuzhoy et al. [13] improve this ratio by exhibiting a randomized $\frac{e}{e-1} + \epsilon$-approximation algorithm for any $\epsilon > 0$. Berman and Dasgupta [5] describe a combinatorial approximation algorithm to deal with the weighted case of the discrete interval scheduling problem, and show how this can be used to derive approximation factors for time-constrained scheduling problems involving identical machines and unrelated machines, improving upon the LP-based approach described by Bar-Noy et al. [4]. Similar bounds in a very general framework are explained by Bar-Noy et al. [3].

3.4 Online interval scheduling problems

Lipton and Tomkins [30] introduce the online interval scheduling problem, where intervals with a given length $f_j - s_j$ are presented to the scheduler in the order of their start time s_j. The scheduler must decide whether to accept each interval before a new interval is presented. The objective is to maximize total length of the accepted intervals while ensuring that no pair of accepted intervals overlap. They show that no (randomized) algorithm can achieve a competitive ratio better than $O(\log \Delta)$ (where Δ denotes the ratio between the longest and the shortest interval), and gave an $O((\log \Delta)^{1+\epsilon})$-competitive algorithm. They also present a 2-competitive algorithm for the case of two lengths.

A related setting where there is a given weight for each interval (not necessarily equal to its length) is considered by Woeginger [34]. Here it is allowed for the scheduler to interrupt a previously accepted interval in order to accept a new interval; the weight of the interrupted interval is then lost. The problem is to maximize the total weight of the accepted (and not interrupted) intervals. Woeginger [34] shows that no deterministic algorithm can achieve a finite competitive ratio.

In case each interval has unit weight, and there are m machines, Faigle and Nawijn [17], and independently Carlisle and Lloyd [10], observed that a greedy algorithm is in fact an online algorithm that always outputs an optimal solution. Erlebach and Spieksma [16] study the online version of

the discrete interval scheduling problem (see Section 3.3) for multiple machines, and they consider the intervals in the order of right endpoints. For this setting they give best possible algorithms for various classes of weight functions.

4 Applications of interval scheduling problems

Interval scheduling problems have a broad diversity of applications, as is witnessed by the following selection (which is not meant as an exhaustive list).

Crew/vehicle scheduling. The basic crew scheduling problem can be formulated as follows. Given is a set of crews, a set of locations, and a set of tasks. For each task a starting time, an ending time and a location is given; for each pair of locations a distance is given. The problem of assigning tasks to crews using a minimum number of crews can be phrased in interval scheduling terminology by viewing the tasks as the intervals, the crews as machines, and allowing for a distance between any pair of intervals. Crew scheduling problems are among the most celebrated problems in operations research. They also appear in bus driver applications (see Fischetti et al. [19] and the references contained therein). A specific problem involving hierarchies is described by Faneyte et al. [18].

Telecommunication. Consider a setting where users communicate with each other using a network. A user communicates by requesting capacity of a link (called bandwidth) in the network during a given interval. Of course, requests of different users may not be compatible due to the amount of capacity requested or the intervals requested. How to utilize the network optimally by allocating the available bandwidth to the users is the main question in this application. Notice that this application is especially relevant in an online context. Using interval scheduling terminology, the requests are viewed as intervals, and links as machines; see Bhatia et al. [6] and the references contained therein for more details.

Other applications. Gabrel [21] considers the following problem in *satellite photography*. A satellite makes orbits around the earth. Its task is to photograph pieces of the earth's surface. Fulfilling a request for photographing a specific piece of the earth implies that the camera needs to start and end filming at given moments in time. When formulated in terms of interval scheduling, a request becomes a job and the camera is the machine.

Anthonisse and Lenstra [1] describe a *cottage rental* problem. Given a set of identical cottages, can a client's request for a reservation for a given time-period immediately be answered? And what if some periods have already been preassigned? Obviously, in an interval scheduling context, a cottage is a machine and a request is an interval. The results described in this paper answer the two questions above.

The interest of Gupta et al. [23] and Hashimoto and Stevens [24] in the basic interval scheduling problem was sparked by a problem in VLSI-layout, called the *channel assignment* problem. Given are pairs of components. Each component must be placed on a printed circuit board at a given y-coordinate. Moreover, components of a pair must be placed on the same x-coordinate. Then, the two components are interconnected using a so-called channel. Channels are not allowed to overlap. Observe now that minimizing the number of channels boils down to solving the basic interval scheduling problem.

Kolen and Kroon [27] discuss a *maintenance* problem in the aviation industry. More specifically, engineers need to carry out maintenance jobs on aircraft. There are different types of aircrafts, and an engineer is only allowed to perform a job on a specific type of aircraft if (s)he has a license for that type. The jobs have fixed starting and ending times. With the licensed engineers in the role of machines, the problem is an interval scheduling problem where each machine can process a

given set of jobs.

Carter and Tovey [11] describe a problem in *class room assignment*. The problem is to assign n classes that come together during given periods to rooms R_j, $j = 1, \ldots, m$, under various constraints and objectives. One of the variants they consider is the setting where the rooms are ordered such that if a class will accept room R_j, then it will also accept any room R_k with $k > j$. With classes in the role of intervals and rooms in the role of machines, a hierarchical interval scheduling problem arises.

Brehob et al. [8] investigate *replacement policies* for non-standard caches (a cache is a piece of memory in a computer; items are stored in cache-locations). They draw a parallel with interval scheduling by letting each access to a certain item correspond to a start time of an interval, and the next access to that item serves as the finishing time of that interval. A machine corresponds to a cache-location.

Another problem comes from *computational biology* (see Chen et al. [12]). Given a sequence of amino-acids (the single machine) and so-called segments (these correspond to the jobs), and a profit for each possible allocation of segment to a position in the sequence of amino-acids, find an allocation such that no two segments overlap, and total weight is maximized.

5 Conclusion

Interval scheduling problems appear in diverse fields ranging from crew scheduling to telecommunication. They form an important class of scheduling problems and have been studied under various names and with application-specific constraints. This presentation reviews different variants of the basic interval scheduling problem; we refer to Kolen et al. [28] for a survey of this topic.

References

[1] J.M. Anthonisse and J.K. Lenstra (1984), Operational operations research at the Mathematical Centre, *European Journal of Operational Research* **15**, 293 – 296.

[2] E.M. Arkin and E.B. Silverberg (1987), *Scheduling with fixed start and end times*, *Discrete Applied Mathematics* **18**, 1 – 8.

[3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J.S. Naor, and B. Schieber. (2005), A unified approach to approximating resource allocation and scheduling, *Journal of the ACM* **48**, 1069 – 1090.

[4] A. Bar-Noy, S. Guha, J.S. Naor, and B. Schieber (2001), Approximating the throughput of multiple machines in real-time scheduling, *SIAM Journal on Computing* **31**, 331 – 352.

[5] P. Berman and B. DasGupta (2000), Multi-phase algorithms for throughput maximization for real-time scheduling, *Journal of Combinatorial Optimization* **4**, 307 – 323.

[6] R. Bhatia, J. Chuzhoy, A. Freund, and J. Naor (2003), Algorithmic aspects of bandwidth trading, Proceedings of the 30-th International Conference on Automata, Languages, and Programming, *Lecture Notes in Computer Science* **2719**, 751–766, Springer, Heidelberg.

[7] K.I. Bouzina and H. Emmons (1996), Interval scheduling on identical machines, *Journal of Global Optimization* **9**, 379 – 393.

[8] M. Brehob, S. Wagner, E. Torng, and R. Enbody (2004), Optimal replacement is NP-hard for nonstandard caches, *IEEE Transactions on Computers* **53**, 73 – 76.

[9] P. Brucker and L. Nordmann (1994), The k-track assignment problem, *Computing* **54**, 97 – 122.

[10] M.C. Carlisle and E.L. Lloyd (1995), On the k-coloring of intervals, *Discrete Applied Mathematics* **59**, 225 – 235.

[11] M.W. Carter and C.A. Tovey (1992), When is the classroom assignment problem hard?, *Operations Research* **40**, S28 – S39.

[12] Chen, Z., G. Lin, R. Rizzi, J. Wen, D. Xu, Y. Xu, and T. Jiang (2005), More reliable protein NMR peak assignment via improved 2-interval scheduling, *Journal of Computational Biology* **12**, 129 – 146.

[13] Chuzhoy, J., R. Ostrovsky, and Y. Rabani (2001), Approximation algorithms for the job interval selection problem and related scheduling problems, *Mathematics of Operations Research* **31**, 730 – 738.

[14] Dantzig, G.B. and D.R. Fulkerson (1954), Minimizing the number of tankers to meet a fixed schedule, *Naval Research Logistics Quarterly* **1**, 217 – 222.

[15] Dondeti, V.R. and H. Emmons (1992), Fixed job scheduling with two types of processors, *Operations Research* **40**, S76 – S85.

[16] T. Erlebach and F.C.R. Spieksma (2003), Interval selection: applications, algorithms, and lower bounds, *Journal of Algorithms* **46**, 27 – 53.

[17] U. Faigle and W.M. Nawijn (1995), Note on scheduling intervals on-line, *Discrete Applied Mathematics* **58**, 13 – 17.

[18] D.B.C. Faneyte, F.C.R. Spieksma, and G.J. Woeginger (2001), A branch-and-price algorithm for a hierarchical crew scheduling problem, *Naval Research Logistics* **49**, 743 – 759.

[19] M. Fischetti, S. Martello, and P. Toth (1992), Approximation algorithms for fixed job schedule problems, Operations Research **40**, S96 – S108.

[20] L.R. Jr. Ford and D.R. Fulkerson (1962), *Flows in networks*, Princeton University Press, Princeton, New Jersey.

[21] Gabrel, V. (1995), *Scheduling jobs within time windows on identical parallel machines: New models and algorithms*, European Journal of Operational Research **83**, 320 – 329.

[22] S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (editors) (1993), Logistics of production and inventory, *Handbooks in Operations Research and Management Science* **4**, North-Holland, Amsterdam.

[23] U.I. Gupta, D.T. Lee, and J.Y.-T. Leung (1979), An optimal solution for the channel-assignment problem, *IEEE Transactions on Computers* **C-28**, 807 – 810.

[24] A. Hashimoto and J. Stevens (1971), Wire routing by optimizing channel assignment within large apertures, *Proceedings of the 8th Design Automation Workshop*, 155 – 169.

[25] Q. Huang and E. Lloyd (2003), Cost constrained fixed job scheduling, Italian Conference on Theoretical Computer Science, *Lecture Notes in Computer Science* **2841**, 111 – 124.

[26] J.M. Keil (1992), On the complexity of scheduling tasks with discrete starting times, *Operations Research Letters* **12**, 293 –295.

[27] A.W.J. Kolen and L.G. Kroon (1994), An analysis of shift class design problems, *European Journal of Operational Research* **79**, 471 – 430.

[28] A.W.J. Kolen, J.K. Lenstra, C.H. Papadimitriou, and F.C.R. Spieksma (2007), Interval Scheduling: A Survey, *Naval Research Logistics*, to appear.

[29] L.G. Kroon, M. Salomon, and L. van Wassenhove (1995), Exact and approximation algorithms for the operational fixed interval scheduling problem, *European Journal of Operational Research* **82**, 190 – 205.

[30] R.J. Lipton and A. Tomkins (1994), Online interval scheduling, *Proceedings of the fifth annual ACM-SIAM Symposium On Discrete Algorithms*, 302 – 311.

[31] K. Nakajima and S.L. Hakimi (1982), Complexity results for scheduling tasks with discrete starting times, *Journal of Algorithms* **3**, 344 – 361.

[32] K. Nakajima, S.L. Hakimi, and J.K. Lenstra (1982), Complexity results for scheduling tasks in fixed intervals on two types of machines, *SIAM Journal on Computing* **11**, 512–520.

[33] F.C.R. Spieksma (1999), On the approximability of an interval scheduling problem, *Journal of Scheduling* **2**, 215 – 227.

[34] G.J. Woeginger (1994), On-line scheduling of jobs with fixed start and end times, *Theoretical Computer Science* **130**, 5 – 16.

Online Stochastic Routing and Scheduling

Pascal Van Hentenryck
Brown University, Box 1910, Providence, RI 02912, USA, pvh@cs.brown.edu

Robotics and Chromatic Scheduling

Dominique de Werra
EPFL, 1015 Lausanne, Switzerland, dewerra.ima@epfl.ch
Marc Demange
ESSEC, F-95021 Cergy Pontoise, France, demange@essec.fr
Tinaz Ekim
EPFL, 1015 Lausanne, Switzerland, tinaz.ekim@epfl.ch

1 Introduction

A classical problem in robotics consists in organizing the moves of a single robot (or of several robots) which have to pick up a collection of items with different sizes along a storage line. A robot may pick up several items during a trip and it has to pile them, which implies that larger items have to be placed below smaller ones: in other words, during a trip a robot has to pick up items in decreasing order of their sizes.
Different hypotheses made on the possible moves of a robot along the storage line (or corridor) or assumptions on the location of the entry/exit points of the corridor will lead to different problems. It turns out that these will correspond to some graph coloring models; it is our objective to derive some of these generalized coloring models and to present some basic results on their complexity.

2 Preliminaries

For graph theoretical definitions not given here, the reader is referred to [1]. In this paper, all graphs are simple and loopless. Given a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$, a subset of pairwise non-adjacent vertices is called a *stable set*, and a subset of pairwise adjacent vertices is called *a clique*.
A problem which is extensively studied in graph theory is *Min Coloring*: it is the problem of minimizing the number of stable sets covering all the vertices of G. Min Coloring has a wide range of applications in scheduling, telecommunication, production systems, etc. In the literature, several generalizations of this problem have also been studied to cover an even wider field of applications. One way of generalizing this usual coloring problem is to use not only stable sets but also cliques to cover all the vertices of a graph. In this paper, we use such generalizations in order to model some problems occurring in automated storage systems. We say that G is (p, k)-*colorable* if its vertex set V can be partitioned into p cliques and k stable sets. Given a graph G, *Min Cocoloring* consists in minimizing the number $p+k$ such that G is (p, k)-colorable. The minimum value is called *cochromatic number* and is denoted by $z(G)$. Another problem, called *Min Split-coloring*, is the one where for a given graph G, we minimize the number $\max(p, k)$ such that G is (p, k)-colorable. The optimal value, called *split-chromatic number*, is denoted by $\chi_S(G)$. The reason why we call this problem split-coloring is that it amounts to partitioning the vertex set of a given graph into a minimum number of split graphs; a graph is a *split graph* if its vertex set admits a partition (S, K), called *split partition*, where S is a stable set and K a clique. Min Split-coloring was first introduced in [8] and studied in [5, 6, 4] which give polynomial time algorithms in restricted classes of graphs and provide approximation algorithms for other cases.

A graph G is a *threshold graph* if it is a split graph with partition (S, K) and the neighborhoods of vertices i in S, denoted by $N(i)$, are nested, i.e., one can label vertices in S with integers such that if $i < j$ then $N(i) \subseteq N(j)$.
In this work, we handle the above generalized coloring problems in permutation graphs; given a permutation π of n numbers, the corresponding *permutation graph* $G(\pi)$ is obtained by representing each number by a vertex and linking vertices i and j with an edge whenever we have $i > j$ and $\pi^{-1}(i) < \pi^{-1}(j)$, $\pi^{-1}(i)$ being the position of number i in π. Note that, in permutation graphs, stable sets correspond to increasing sequences and cliques to decreasing sequences. Partitioning permutations into increasing and decreasing sequences has been extensively studied [14, 12, 2]. Our work will extend this research and raise some related questions. Let us first mention the preliminary results on the complexity of Min Split-coloring.

Theorem 2.1 ([4]) *Let $\mathcal{G}$ be a class of graphs closed under addition of cliques without link to the rest of the graph and under addition of stable sets completely linked to the rest of the graph; then for any graph in $\mathcal{G}$ Min Cocoloring reduces to Min Split-coloring.*

Corollary 2.2 *Min Split-coloring is NP-hard in permutation graphs.*

The corollary follows from Theorem 2.1 and the fact that Min Cocoloring is NP-hard in permutation graphs [14].

3 Various models of pick up robots in an automated storage system

We shall consider several situations where a robot has to pick up a collection of items with different sizes which are located in a storage shelf. One may see the storage shelf as a predefined trajectory that the robot can cross in both directions. For simplicity purposes we will assume that the shelf has just one level so that the items (characterized by their size) are located in line along a storage corridor.
In addition, we assume that the items are labelled: $1, 2, \ldots, n$ in the order of their decreasing sizes. So their positions along the corridor are given by a permutation π of $\{1, 2, \ldots, n\}$. The robot is able to pick up several items but these should be arranged in a pile. For obvious stability reasons when a pile is constructed one is never allowed to put an item j on top of an item i if $j < i$, i.e., if j has a larger size than i. So the robot picks up items in increasing order of their labels. Note that, in a more general framework, the labels may correspond to any order, due to some precedence constraints, in which some tasks should be executed by the robot. In this paper, all the models are illustrated for the case where the order is determined by the requirement of decreasing sizes of the items to be collected.
We shall describe six situations which will lead to different graph theoretical models. The storage corridor will have a left end called L and a right end called R. In all of these models, the unloading operations performed at an Entry/Exit station E cause some idle time for the robots. For this reason, we want to minimize the number of times the robot has to be unloaded.
In each model, continuous arrows represent the possible exit trajectories from the Entry/Exit station E, whereas dashed arrows represent the possible entrance trajectories to E. We also show the possible moves of the robot along the corridor by (broken) arrows depicted under the corridor.

3.1 One-directional system

Here there is an Entry/Exit station E which is outside of the corridor; all robot trips are from L to R then to E and back to L for a new trip.
If we call $G(\pi)$ or simply G the permutation graph associated to π, then during a trip the robot will be able to collect some items in increasing order of their labels; the vertices corresponding to these items will form a stable set in G. So, minimizing the number of trips made by a robot for collecting all items amounts to finding a usual coloring of a permutation graph G. In other words one has to cover π with a minimum number of increasing sequences.

3.2 Bi-directional system with a unique E

The robot is now allowed to have trips from L to R and then to E as well as trips from R to L and then to E. In other words, whenever a robot is at E, it may go to L or to R to start its trip along the storage corridor.
Each trip of the robot will be represented by an increasing or a decreasing sequence in π. Minimizing the number of trips required to pick up all items corresponds to finding an optimum cocoloring of G, i.e., a partition into a minimum number of stable sets and cliques: each trip corresponds in π to a monotone sequence, that is either an increasing sequence (trip from L to R) or a decreasing sequence (trip from R to L), i.e., to a stable set or to a clique.

3.3 Bi-directional system with E at both ends

In this case we have at each end L and R an Entry/Exit station E. The robot may start from one end, pick up items while moving to the other end, then unload its pile and possibly start a trip back to the end where it entered the corridor. The items collected during each trip from L to R will correspond to a stable set in G (increasing sequence) and for each trip from R to L the items collected will define a clique. So all the items collected during a trip from L to R and then from R to L (or the inverse) form a split graph. We may assume that after the last trip, the robot may possibly make a non-collecting trip to get back to the station where it started. Hence minimizing the number of two-way trips of the robot will be a split-coloring problem; it will consist of partitioning the vertex set of G into a minimum number of split graphs.

3.4 Two-way trips from L

We shall assume now that at end L of the storage corridor there is an Entry/Exit station. The robot may now enter the corridor at L, then it will move to the other end R and come back where it started so that it can unload the collected items.
In the permutation π, each trip (starting from L) will be represented by an increasing sequence $a_1 < a_2 < \cdots < a_p$ (trip from L to R) followed by an increasing sequence $a_{p+1} < \cdots < a_r$ with $a_{p+1} > a_p$ (since this last part corresponds to the trip from R to L, it will be in fact decreasing in π and it will be represented by a clique in G). The items collected in such a two-way trip will form a threshold graph in G; this can be seen as follows: if j is collected after i during the trip back to L, then j comes before i in π and $j > i$, so j will be linked to all neighbors of i which are after i in π. Hence $N(j) \supseteq N(i)$.
So in this case, if one can pick up all items with l two-way trips all from L, then there are l threshold graphs which cover all vertices of G. The links of this problem with graphs will be discussed in more details in Section 5. Note that a symmetrical model is obtained if the Entry/Exit station is located at R instead of L.

3.5 Zigzag trips

It is reasonable to extend the above type of models by assuming that the robot may move back and forth several times in the corridor. This can be allowed, for instance, by the fact that the robot receives some fixed energy impulse at each visit of the Entry/Exit station which is sufficent to execute such a multiple trip. If the robot makes l trips (alternating trips from L to R and from R to L) before leaving the corridor, the items collected will be represented by a so-called $(l-1)$-modal sequence; it can be seen as a trip where the robot changes direction $l-1$ times.

3.6 Two-way trips from either end

Here, we focus on a special case of model 3.5. The robot collects items with two-way trips (meaning that the items are unloaded only at the end of a back and forth trip) from either L or R. The possibility of doing two-way trips is provided by the energy impulse given at each visit of the Entry/Exit station. The location of the Entry/Exit station allows the robot to execute two-way trips that can start from either end; such a trip in this model is called *feasible trip*. It is easy to see that a feasible trip is obtained by merging a sequence $a_1 < a_2 < \ldots < a_s$ and $b_1 > b_2 > \ldots > b_t$ with either $a_s < b_t$ (robot starting from L), or $b_1 < a_1$ (robot starting from R). It follows from the explanations given in model 3.4 that the items collected during each feasible trip in this model induces a threshold graph in the corresponding permutation graph G. The problem of covering a given permutation π (or equivalently collecting all the items represented by π) by a minimum number of feasible trips will be called *Min Ordered Collecting*, and its optimal value will be denoted by $\rho(\pi)$. Also, if we denote by $\rho_l(\pi)$ (respectively $\rho_r(\pi)$) the values of optimal ordered collections where all feasible trips start from left (respectively right) as described in model 3.4, then clearly $\rho(\pi) \leq \min(\rho_l(\pi), \rho_r(\pi))$. The links of Min Ordered Collecting with graphs will be discussed in Section 5.

4 Some results on related problems

We first introduce the necessary definitions and tools to study the models described above. Then, polynomial time algorithms will be derived for some special cases.

4.1 Terminology and NP-hardness

Before mentioning the complexity status of Min Ordered Collecting, let us define the notion of l-modal sequences, introduced in [13].
We call *internal extremum* of a permutation π, a number $\pi(i)$ such that $2 \leq i \leq n-1$ and we have either $\pi(i) < \pi(i-1)$ and $\pi(i) < \pi(i+1)$, or $\pi(i) > \pi(i-1)$ and $\pi(i) > \pi(i+1)$. A sequence is *l-modal* if it has at most l internal extrema, the first being of either type. If the first extremum is a maximum then the sequence is called *upper l-modal* and otherwise *lower l-modal*. In the case $l = 1$, we say that a sequence is (upper or lower) *unimodal*. Finally, if a permutation can be partitioned into p decreasing and k increasing subsequences then it is called *p-decreasing k-increasing*.
Let $\pi = (\pi(1), \ldots, \pi(n))$ be a permutation, the *reversal* of π, denoted by $\bar{\pi}$, is the permutation $(\pi(n), \ldots, \pi(1))$. The *inverse permutation*, denoted by π^{-1}, is given by $(\pi^{-1}(1), \ldots, \pi^{-1}(n))$. Finally, we call the *symmetric* of π, the permutation $\pi_s = (n+1-\pi(1), \ldots, n+1-\pi(n))$. Clearly, $\rho_l(\pi) = \rho_r(\bar{\pi})$ and $\rho_r(\pi) = \rho_l(\bar{\pi})$. One can notice that ρ is an invariant with respect to the symmetric and the reversal operations, i.e., $\rho(\pi) = \rho(\bar{\pi}) = \rho(\pi_s)$, since both interchange increasing and decreasing subsequences. Note also that $\rho_l(\pi) = 1$ if and only if π^{-1} is upper unimodal, and $\rho_r(\pi) = 1$ if and only if π^{-1} is lower unimodal; moreover $(\pi^{-1})^{-1} = \pi$. Finally, it can be easily

checked that π is upper l-modal if and only if π_s is lower l-modal and vice versa. To illustrate, consider the permutation $\pi = (1, 5, 2, 4, 3)$ which is a feasible trip from left; then $\pi^{-1} = (1, 3, 5, 4, 2)$ is upper unimodal and $(\pi^{-1})_s = (5, 3, 1, 2, 4)$ is lower unimodal.
Using the above terminology, one can state that minimizing feasible trips covering a given permutation π is equivalent to finding the minimum k such that there are at most k unimodal subsequences covering π^{-1}.
Now, we assume that for reasons of autonomy the robot is constrained to make at most $l + 1$ trips back and forth along the corridor before going to the Entry/Exit station, as introduced in model 3.5. It is therefore reasonable to try to minimize the number of times the robot has to make a trajectory with (at most) $l + 1$ trips back and forth. This amounts to covering the permutation π representing the items in the corridor by a minimum number of l-modal sequences.
Let us call *Min l-modal*, *Min upper l-modal* and *Min unimodal* the problems of covering a given permutation with a minimum number of respectively l-modal, upper l-modal and unimodal subsequences, then the following theorem of [13] establishes the NP-hardness of all these problems.

Theorem 4.1 ([13]) *Min l-modal, Min upper l-modal and Min unimodal are NP-hard even for fixed l, in particular for $l = 1$.* □

Corollary 4.2 *Given a permutation π, it is NP-hard to find $\rho(\pi), \rho_l(\pi)$ and $\rho_r(\pi)$.*

Proof 4.1 *Given a permutation π, feasible trips from left (respectively right) in π become upper (respectively lower) unimodal sequences in π^{-1}; moreover the inverse permutation is obtained in polynomial time.* □

In what follows, we formulate our results in terms of Min (upper/lower) l-modal problem rather than feasible trips. A sequence that can be covered by k (upper/lower) l-modal subsequences will be called *k-(upper/lower) l-modal* and similar notations hold for the unimodal case.

4.2 Recognition of 2-lower unimodal permutations

This would correspond to the case where all items could be picked up by two trips of the robot starting from R and making two-way trips (See model 3.4).

The following result simply states that there is a polynomial time algorithm to decide whether the items in the corridor can be picked up by two trips of a robot starting from R and making two-way trips.

Theorem 4.3 *There is an algorithm that decides in time $O(m + n \log n)$ whether a given permutation π is 2-lower unimodal or not.*

By symmetry, if the robot starts from L, we can clearly state:

Corollary 4.4 *Given a permutation π, it can be decided in time $O(m + n \log n)$ whether it is 2-upper unimodal or not.* □

4.3 Maximum l-modal subsequence

Consider a permutation π which is upper or lower l-modal. Then, it can be easily checked that π^{-1} corresponds to an increasing subsequence of labels representing items collected during $l + 1$ trips back and forth of a robot (starting from left and right, respectively) which does not unload the collected items until the end of $l + 1$ trips and which obeys the constraint of increasing labels during its whole trip (see model 3.5). In other words, a robot can collect all the items ordered

according to π^{-1} with $l+1$ back and forth trips respecting the constraint of increasing labels. In what follows, given a permutation π representing the sizes of the items, we show how to collect a maximum number of items by $l+1$ trips of a robot. This will be explained in terms of maximum l-modal subsequence of a given permutation.

We construct from a permutation π a permutation π^l such that there is a one-to-one correspondence between the upper l-modal subsequences in π and the increasing subsequences in π^l. Since a maximum increasing subsequence in π^l can be found in time $O((l+1)n\log((l+1))n))$ we get:

Theorem 4.5 *Given a permutation π, a maximum upper or lower l-modal subsequence of π can be found in time $O(n\log n)$ for fixed l, and in time $O(n^2\log n)$ for arbitrary l.*

5 Min Ordered Collecting vs. Min Threshold-coloring

In this section, we explore the links between feasible trips (or unimodal subsequences) and split graphs. We recall that feasible trips correspond in fact to a subclass of split graphs, namely the threshold graphs. Nevertheless, as will be explained below, a minimum ordered collecting ρ is a parameter defined in permutations and not in permutation graphs. These results give birth to the idea of considering the problem of covering the vertices of a given graph with a minimum number of threshold graphs.
The following notion is given in [11].

Definition 5.1 (shuffle product) *Let σ and τ be two sequences. The shuffle product of σ and τ is defined as follows:*

$$\sigma \sqcup\!\sqcup \tau = \{\sigma_1\tau_1 \dots \sigma_k\tau_k : \sigma = \sigma_1 \dots \sigma_k \text{ and } \tau = \tau_1 \dots \tau_k\}$$

where σ_i and τ_i are subsequences, k ranges over all integers and the juxtaposition means concatenation.

Now, let us observe the following fact.

Proposition 5.1 *Let π be a permutation, then:*

$$\begin{aligned} \rho_l(\pi) = 1 &\Leftrightarrow \pi \text{ is a shuffle product of type 1} : [1,\dots,p]\sqcup\!\sqcup[n,\dots,p+1], 1 \le p < n; \\ \rho_r(\pi) = 1 &\Leftrightarrow \pi \text{ is a shuffle product of type 2} : [p+1,\dots,n]\sqcup\!\sqcup[p,\dots,1], 1 \le p < n; \end{aligned}$$

where shuffle products can also have only one term.

The relationship between threshold graphs and feasible trips is derived from the following theorem of [10].

Theorem 5.2 ([10]) *The threshold graphs are precisely those permutation graphs corresponding to permutations contained in*

$$[1,2,\dots,p]\sqcup\!\sqcup[n,n-1,\dots,p+1]$$

where p and n are positive integers. □

A natural extension of Min Ordered Collecting in graphs appears as the following problem. These extensions may have no obvious interpretation in terms of item collection by robots along a storage corridor; we give them anyway for their mathematical interest.

Definition 5.3 (Min Threshold-coloring) *Min Threshold-coloring is the problem of covering the vertices of a given graph G by a minimum number of threshold graphs. The optimal value, called threshold-chromatic number, is denoted by $\chi_T(G)$.*

Theorem 5.4 *Let $\mathcal{G}$ be a class of graphs closed under addition of cliques without link to the rest of the graph and under addition of stable sets completely linked to the rest of the graph, then for any graph in $\mathcal{G}$ Min Cocoloring reduces in polynomial time to Min Threshold-coloring.*

Corollary 5.5 *Min Threshold-coloring is NP-hard in permutation graphs.*

6 Open questions and discussion

There are several questions to explore on the links between Min Ordered Collecting and Min Threshold-coloring. In view of Theorem 5.2 and the following discussion, we have clearly $\rho(\pi) \geq \chi_T(G(\pi))$ for any permutation π. It would be interesting to know, given a permutation graph G, whether there is a permutation π such that $G(\pi)$ is isomorphic to G and $\rho(\pi) = \chi_T(G)$. If this is the case, then an optimal threshold coloring of G would give an optimal ordered collecting of the items forming the corresponding permutation π. Also, one could study, for a permutation graph G, how large can be the quantity $\max_{\pi_i}(\rho(\pi_i) - \chi_T(G))$, where $G(\pi_i)$ is isomorphic to G.
Another interesting topic concerns permutations for which an optimal ordered collecting is a solution where the robot departs always from the same end. In other words, for such permutations, locating the Entry/Exit station outside of the corridor does not provide any advantage over the more constrained model where the Entry/Exit station is fixed to one end of the corridor. This notion yields open questions of high mathematical interest.

Definition 6.1 (ideal permutation) *A permutation π is called ideal if we have $\rho(\pi) = \min(\rho_l(\pi), \rho_r(\pi))$.*

Definition 6.2 (minimal obstruction) *A permutation π is a minimal obstruction if it verifies $\rho(\pi) < \min(\rho_l(\pi), \rho_r(\pi))$ and all subpermutations $\pi' \subset \pi$ are ideal, i.e., $\rho(\pi') = \min(\rho_l(\pi'), \rho_r(\pi'))$.*

In [9], lower and upper bounds on the size of minimal obstructions for permutations having $\rho(\pi) = 2$ is studied. It can be easily shown by case enumeration that all permutations of size less than or equal to 7 are ideal; smallest minimal obstructions that are detected have size 8. For instance, $\pi = (1, 8, 4, 3, 6, 2, 7, 5)$ is a minimal obstruction; $\rho(\pi) = 2$ where the robot collects the items with labels $(1, 5, 8)$ by starting from left, and the items with labels $(4, 3, 6, 2, 7)$ by starting from right. However, it can be easily checked that $\rho_l(\pi) = \rho_r(\pi) = 3$ and that for all subpermutations $\pi' \subset \pi$, we have $\rho(\pi') = \min(\rho_l(\pi'), \rho_r(\pi'))$. On the other hand, it is also shown by computer enumeration that obstructions of size 9 always contain an obstruction of size 8, that is, there is no minimal obstruction of size 9. These results allow us to state the following conjecture.

Conjecture 6.3 *All minimal obstructions π with $\rho(\pi) = 2$ are of size 8.*

A natural extension of the robotics problems considered in this paper is the case where the arrangement of the items on the storage corridor is not known in advance, but it is revealed in time. The study of such on-line models would be of great use in practice; in fact it is natural to think that items are delivered at different times depending on the rest of the production system, and that one has to decide the moves of the robot each time an item is delivered to the storage system. We can already find in [13] and [7] some results on the on-line version of the model 3.2; further research is needed on other models.

References

[1] C. Berge (1976), *Graphs and Hypergraphs*. North Holland Publishing Company.

[2] A. Brandstädt and D. Kratsch (1986), On partitions of permutations into increasing and decreasing subsequences. *Journal of Information Processing and Cybernetics* **22**(5/6), 263 – 273.

[3] V. Chvátal (1984), Perfectly ordered graphs. In *Topics on Perfect Graphs (C. Berge and V. Chvátal eds.)* **21**, *Annals of Discrete Mathematics*, 63 – 65.

[4] M. Demange, T. Ekim, and D. de Werra (in press), On the approximation of Min Split-coloring and Min Cocoloring. *Journal of Graph Algorithms and Applications*.

[5] M. Demange, T. Ekim, and D. de Werra (2005), Partitioning cographs into cliques and stable sets. *Discrete Optimization* **2**, 145 – 153.

[6] M. Demange, T. Ekim, and D. de Werra (2005), (p, k)-coloring problems in line graphs. *Theoretical Computer Science* **349**(3), 462 – 474.

[7] M. Demange and B. Leroy-Beaulieu (2006), Online Coloring of Comparability Graphs: some results. http://infoscience.epfl.ch/search.py?recid=102343, *Ecole Polytechnique Fédérale de Lausanne (ROSE)*.

[8] T. Ekim and D. de Werra (2005), On split-coloring problems, *Journal of Combinatorial Optimization* **10**, 211 – 225.

[9] F. Gilliéron (2006), Collecte d'objets par ordre décroissant de leur taille, Semester project, *Ecole Polytechnique Fédérale de Lausanne (ROSE)*.

[10] M. C. Golumbic (1978), Threshold graphs and synchronizing parallel processes, **18** of *Combinatorics*, Colloq. Math. Soc. Janos Bolyai, North Holland, Budapest, 419 – 428.

[11] M. C. Golumbic (1980), *Algorithmic graph theory and perfect graphs*, Computer Science and Applied Mathematics, Academic Press.

[12] A.E. Kézdy, H.S. Snevily, and C. Wang(1996), Partitioning permutations into increasing and decreasing subsequences, *Journal of Combinatorial Theory Series A* **73**(2), 353 – 359.

[13] G. Di Stefano, S. Krause, M.E. Lübbecke, and U.T. Zimmermann (2006), On minimum k-modal partitions of permutations, In *LATIN 2006: Theoretical informatics* **3887** of *Lecture Notes in Comput. Sci.* Springer, Berlin, 374 – 385.

[14] K. Wagner (1984), Monotonic coverings of finite sets, *Elektron. Inf. Kybern. EIK* **20**(12), 633 – 694.

Part II.

Regular Papers

Semi On-Line Scheduling on Two Uniform Processors

Enrico Angelelli, Maria Grazia Speranza
Università di Brescia, C.da S.Chiara 50 - Brescia, Italy, {angele, speranza}@eco.unibs.it
Zsolt Tuza
Computer and Automation Institute, Hungar. Acad. Sci., Budapest, Hungary, tuza@sztaki.hu
Department of Computer Science, University of Pannonia, Veszprém, Hungary

In this paper we consider the problem of semi on-line scheduling on two uniform processors, in the case where the total sum of the tasks is known in advance. Tasks arrive one at a time and have to be assigned to one of the two processors before the next one arrives. The assignment cannot be changed later. The objective is the minimization of the makespan. We derive lower bounds and algorithms, depending on the value of the speed s of the fast processor (the speed of the slow processor is normalized to 1). The algorithms presented for $s \geq \sqrt{3}$ are optimal, as well as for $s = 1$ and for $\frac{1+\sqrt{17}}{4} \leq s \leq \frac{1+\sqrt{3}}{2}$.

Keywords: Multi-processor Scheduling, Real-Time Scheduling, Theoretical Scheduling.

1 Introduction

The typical framework of *on-line* optimization problems requires to take decisions as data becomes available over time, while no information is given about the future. When all information is available at one time before optimization, the problem is called *off-line*. In on-line scheduling, tasks arrive one by one and no information is available about the number or the size of tasks that will arrive in the future. As soon as a task arrives it has to be assigned to a processor and the decision cannot be changed later. This framework captures an important feature of real problems, the absence of information. Nevertheless, even in real applications some information may be available to the decision makers. In this case the problem is called *semi on-line*. Though optimization algorithms for semi on-line problems cannot be as effective as those designed for off-line frameworks, even a little additional information may allow a dramatic improvement of the performance with respect to the pure on-line setting. Thus, it becomes of relevant interest to study the impact of information on the performance of algorithms for a given problem.

1.1 Problem definition and notation

In this paper we consider the semi on-line scheduling problem on two uniform processors in the case where the sum of the tasks is known in advance.

A set of ordered tasks $\{p_1,\ p_2,\ \ldots,\ p_n\}$ such that $\sum_{j=1}^{n} p_j = 1 + s$ arrive one at a time and have to be immediately assigned to one of two processors. The speed of the slower processor is normalized to 1, and the speed of the faster one is denoted by $s \geq 1$; as a particular case, for $s = 1$ we have two identical processors. We call the processors P_1 and P_s, referring in the subscript to their speed. The processing time of task p_j on processor P_h is p_j/h.

Given a problem instance, we indicate by L_1^i and L_s^i the loads of the processors just before the assignment of task p_i and by L_1 and L_s the final loads of the two processors. The objective of the problem is to minimize the makespan $\max(L_1, L_s/s)$. Moreover, we use H to indicate both an

algorithm and the makespan generated by the same algorithm on the given instance. The optimum off-line makespan is denoted by Z^*.

The performance of an on-line algorithm is measured by the *competitive ratio*. An on-line algorithm H for a minimization problem is said to be r-competitive if for any instance, given the off-line optimum Z^*, the inequality $H \leq r \cdot Z^*$ holds. The competitive ratio R_H is defined as $\inf\{r \mid H \text{ is } r\text{-competitive}\}$. An algorithm H is said to be *optimal* if no other algorithm has a better competitive ratio (see Sleator and Tarjan [12]).

1.2 Literature

The (semi) on-line scheduling problem where a sequence I of tasks has to be assigned to a set of m processors so that the maximum completion time (*makespan*) is minimized has attracted the interest of many researchers. There are several variants for this problem. For example, the processors may be identical or uniform, while the scheduling may be preemptive or non-preemptive. There is also a number of different algorithmic approaches, deterministic and randomized algorithms, algorithms designed for a fixed number of processors and algorithms for any number of processors. We refer to [11] for a recent survey on (semi) on-line scheduling problems.

For the pure on-line problem, the well-known *List* algorithm introduced by Graham [8] has been proved to be the best possible for the case of identical processors. The *List* algorithm assigns the incoming task to the least loaded processor; and its natural extension to uniform processors assigns the task to the processor where it can be completed first. Cho and Shani [4] proved that *List* is $\frac{1+\sqrt{5}}{2}$-competitive for all s and the bound is tight when $s = \frac{1+\sqrt{5}}{2}$. Epstein *et al.* [5] extended this result to show that *List* is $\min\{\frac{2s+1}{s+1}, 1+\frac{1}{s}\}$-competitive and optimal for all speeds s. Epstein *et al.* [5] provided also randomized algorithms with better performance than *List* in the case that no preemption is allowed. In the same paper, for the problem with preemption, an optimal $1+s/(s^2+s+1)$-competitive algorithm for all $s \geq 1$ was proposed that cannot be beaten by any randomized algorithm. This latter result was obtained independently by Wen and Du [13]. Several semi on-line scheduling problems have been studied for the case of two processors, both identical and uniform. For identical processors, the case of given sum of the tasks and a number of variants were studied in [10], [9], [1], [2] and [3]. For the case of two uniform processors, Epstein and Favrholdt proposed optimal semi on-line algorithms on two uniform processors when tasks arrive with non-increasing size, in both the non-preemptive case [6] and the preemptive case [7].

1.3 Our results

In this paper we provide both lower and upper bounds for the problem. All possible values of speed s for the fast processor are taken into account. For some ranges of s the proposed algorithms are proved to be optimal. Namely, the first algorithm is optimal with competitive ratio 4/3 for the special case $s = 1$ (identical processors), the second algorithm is optimal with competitive ratio s in the range $\frac{1+\sqrt{17}}{4} \leq s \leq \frac{1+\sqrt{3}}{2}$ and, finally, the third algorithm is optimal for all $s \geq \sqrt{3}$ with competitive ratio $1+\frac{1}{s+1}$. Moreover, we show that, despite the fact that in many semi on-line problems the *List* algorithm implicitly exploits the available information and performs better than in the pure on-line setting (see [6] and [9] for examples), in this case there is a need for specialized algorithms. Indeed, when the total size of the tasks is fixed, the *List* algorithm does not improve the bound $\min\{\frac{2s+1}{s+1}, 1+\frac{1}{s}\}$ showed in [5]. Furthermore, we note that, for $s \geq \frac{1+\sqrt{5}}{2}$, the worst-case ratio for *List* is the same as the one obtained by the trivial algorithm that blindly assigns all tasks to the fast processor.

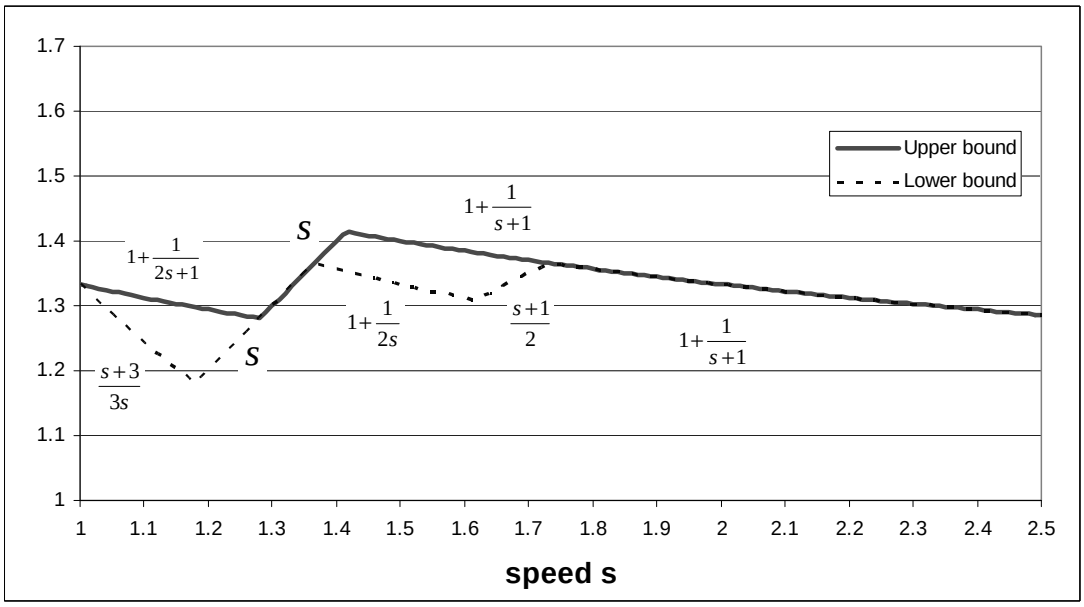


Figure 1: Bounds for the semi on-line problem.

In Section 2 we prove lower bounds for the problem and in Section 3 we propose three algorithms and prove their competitive ratio.

In Figure 1 we show the structure of the largest lower bounds and the best upper bounds we obtained depending on the value of the speed s. In Figure 2 we show the comparisons of the best upper bound for the semi on-line problem versus the best lower bound for the pure on-line problem. This proves the effectiveness of the information made available.

2 Lower bounds

In this section we present a number of lower bounds as functions of the speed s. The lower bounds are presented according to the range of speed s for which they represent the best lower bound. Though, in each theorem statement we claim the full range of speed for which the lower bound holds.

In order to derive lower bounds on the algorithms for the problem, we define a number of parametric instances which we describe through the size of the tasks listed in parentheses in the order of arrival:

$A = \{\delta, \delta, s - 2\delta, 1\}$
$B = \{\delta, \delta, s - \delta, 1 - \delta\}$
$C = \{\delta, \delta, s, 1 - 2\delta\}$
$D = \{\delta, s - \delta, 1\}$

The instances depend on the parameter δ such that $\delta \leq \frac{1}{2}$. The off-line optimum is equal to 1 for all the instances above.

In order to simplify the derivation of the lower bounds, it is convenient to consider the following classes of algorithms:

$H_{10} \equiv \{$algorithms assigning task $p_1 = \delta$ to $P_1\}$

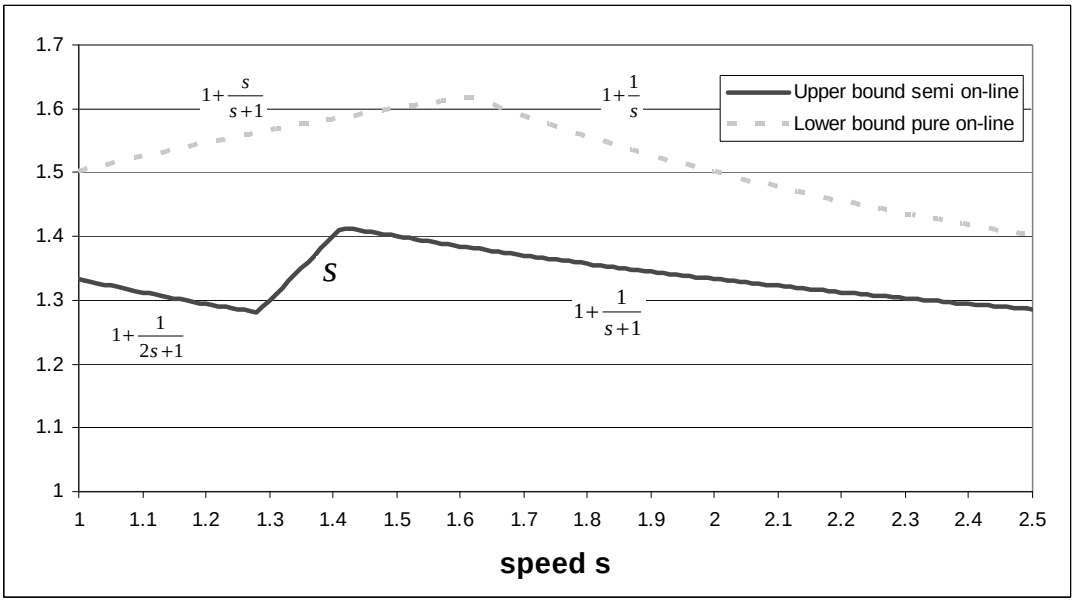


Figure 2: Semi on-line vs pure on-line problem.

$H_{20} \equiv \{$algorithms assigning tasks $p_1, p_2 = \delta$ to $P_1\}$
$H_{02} \equiv \{$algorithms assigning tasks $p_1, p_2 = \delta$ to $P_s\}$
$H_{11} \equiv \{$algorithms assigning tasks $p_1, p_2 = \delta$ to P_1 and P_s, in any order$\}$.

The following relations between the set H of all algorithms and the above classes of algorithms will be used:

$H = H_{10} \cup H_{11} \cup H_{02}$
$H = H_{11} \cup H_{20} \cup H_{02}$.

With a little abuse of notation we indicate with $H_{10}(D)$ any general lower bound that is valid for the makespan obtained by all algorithms in the set H_{10} on instance D. We use a similar notation for all the other classes of algorithms and instances. We will use the following bounds:

$H_{10}(D) \geq \min(1+s, \frac{1+s-\delta}{s}, s, 1+\delta)$
$H_{11}(A) \geq \min(1+s-\delta, \frac{1+s-\delta}{s}, \max\left(\frac{\delta+1}{s}, s-\delta\right), 1+\delta)$
$H_{11}(C) \geq \min(1+s-\delta, \frac{1+s-\delta}{s}, \delta+s, \frac{\delta+s}{s})$
$H_{20}(B) \geq \min(1+s, \frac{1+s-2\delta}{s}, s+\delta, 1+\delta)$
$H_{02}(B) \geq \min(1+s-2\delta, \frac{1+s}{s}, \max(s-\delta, \frac{1+\delta}{s}), \frac{s+\delta}{s})$
$H_{02}(C) \geq \min(1+s-2\delta, \frac{1+s}{s}, s, \frac{s+2\delta}{s})$.

We are now ready to derive the lower bounds on any algorithm for the solution of the semi on-line problem on two processors.

2.1 The range $1 \leq s < \frac{1+\sqrt{37}}{6} \cong 1.1805$

Theorem 1 *No algorithm can guarantee a competitive ratio better than* $\frac{s+3}{3s}$ *for* $s \in [1, \frac{3}{2}]$.

Proof. Let us fix $\delta = \frac{s}{3}$ and view the set of algorithms H as $H = H_{11} \cup H_{20} \cup H_{02}$. The lower bound is implied by the fact that $H_{11}(C)$, $H_{20}(B)$ and $H_{02}(B)$ are bounded from below by $\frac{s+3}{3s}$. □

2.2 The range $\frac{1+\sqrt{37}}{6} \leq s < \frac{1+\sqrt{3}}{2}$

Theorem 2 *No algorithm can guarantee a competitive ratio better than s for $s \in [1, \frac{1+\sqrt{3}}{2}]$.*

Proof. Let us fix $\delta = (s^2 - s) \leq \frac{1}{2}$, for $s \in [1, \frac{1+\sqrt{3}}{2}]$, and view the set of algorithms H as $H = H_{10} \cup H_{11} \cup H_{02}$. The lower bound is implied by the fact that $H_{10}(D)$, $H_{11}(C)$ and $H_{02}(C)$ are bounded from below by s. □

2.3 The range $\frac{1+\sqrt{3}}{2} \leq s < \frac{1+\sqrt{5}}{2}$

Theorem 3 *No algorithm can guarantee a competitive ratio better than $1 + \frac{1}{2s} = \frac{2s+1}{2s}$ for $s \in [\frac{1+\sqrt{3}}{2}, 2]$.*

Proof. Let us fix $\delta = \frac{1}{2}$, for $s \in [\frac{1+\sqrt{3}}{2}, 2]$, and view the set of algorithms H as $H = H_{10} \cup H_{11} \cup H_{02}$. The lower bound is implied by the fact that $H_{10}(D)$, $H_{11}(C)$ and $H_{02}(C)$ are bounded from below by $\frac{2s+1}{2s}$. □

2.4 The range $\frac{1+\sqrt{5}}{2} \leq s < \sqrt{3}$

Theorem 4 *No algorithm can guarantee a competitive ratio better than $\frac{s+1}{2}$ for $s \in [1, \sqrt{3}]$.*

Proof. Let us fix $\delta = \frac{s-1}{2}$, for $s \in [1, \sqrt{3}]$, and view the set of algorithms H as $H = H_{10} \cup H_{11} \cup H_{02}$. The lower bound is implied by the fact that $H_{10}(D)$, $H_{11}(A)$ and $H_{02}(C)$ are bounded from below by $\frac{s+1}{2}$. □

2.5 The range $\sqrt{3} \leq s$

Theorem 5 *No algorithm can guarantee a competitive ratio better than $1 + \frac{1}{s+1} = \frac{s+2}{s+1}$ for $s \geq \sqrt{3}$.*

Proof. The proof is straightforward from the following two Lemmas. □

Lemma 6 *No algorithm can guarantee a competitive ratio better than $\frac{s+2}{s+1}$ for $s \in [\sqrt{3}, 2]$.*

Proof. Let us fix $\delta = \frac{1}{s+1}$, for $s \in [\sqrt{3}, 2]$, and view the set of algorithms H as $H = H_{10} \cup H_{11} \cup H_{02}$. The lower bound is implied by the fact that $H_{10}(D)$, $H_{11}(A)$ and $H_{02}(C)$ are bounded from below by $\frac{s+2}{s+1}$. □

Lemma 7 *No algorithm can guarantee a competitive ratio better than $\frac{s+2}{s+1}$ for $s \geq 2$.*

Proof. Let us fix $\delta = \frac{1}{s+1}$ and consider the following instance, where the incoming task is generated according to the situation of the loads of the two processors:

a) if $L_1^i < \delta$ and $L_s^i < 1 - \delta$, then $p_i = \min\{\delta - L_1^i, (1 - \delta) - L_s^i\}$;

b) if $L_1^i = \delta$, then $p_i = 1$, $p_{i+1} = s - (\delta + L_s^i)$ and p_{i+1} is the last task;

c) if $L_s^i = 1 - \delta$, then $p_i = s$, $p_{i+1} = \delta - L_1^i$ and p_{i+1} is the last task.

According to rule a), both inequalities $L_1^i + p_i \leq \delta$ and $L_s^i + p_i \leq 1 - \delta$ hold, and after the assignment at least one of the two inequalities is strict and the sum of the loads of the two processors is still less than 1.

If the instance ends in case b), then $p_{i+1} > 1$ because $s \geq 2$ and the off-line optimum is equal to 1. The algorithm cannot obtain a value better than

$$\min\{1+\delta, \frac{L_s^i + 1 + s - (\delta + L_s^i)}{s}\} = \min\{1+\delta, \frac{1+s-\delta}{s}\} = \frac{s+2}{s+1}.$$

If the instance terminates in case c), then the off-line optimum is equal to $\frac{s}{s} = 1$ while the algorithm cannot obtain a value better than

$$\min\{s + L_1^i, \frac{1+s-\delta}{s}\} \geq \min\{s, \frac{1+s-\delta}{s}\} = \min\{s, \frac{s+2}{s+1}\}.$$

In conclusion, the algorithm cannot obtain a ratio better than $\frac{s+2}{s+1}$ when $s \geq 2$. □

3 Algorithms

In this section we first show that the *List* algorithm does not improve its performance when the total sum of the tasks is fixed with respect to the pure on-line problem, and then we present three different algorithms that in some intervals of the values of the speed s are optimal. We classify the algorithms according to the values of speed s for which they represent the best lower bound. Though, in each theorem statement, we claim the full range of speed for which the competitive ratio holds.

Proposition 8 *When the sum of the tasks is fixed, the List algorithm has competitive ratio* $\min[1+\frac{s}{s+1}, 1+\frac{1}{s}]$.

Proof. It is already known that $\min[1+\frac{s}{s+1}, 1+\frac{1}{s}]$ is an upper bound for the performance of *List* algorithm in pure on-line setting. Thus, the bound holds for all particular cases like the fixed sum of the tasks. What we need to show is that *List* cannot guarantee a better performance. It is enough to consider the two instances:
$I = \{\frac{s^2}{s+1}, (1-\frac{s^2}{s+1}), s\}$ when $s < \frac{1+\sqrt{5}}{2}$ and
$I = \{1, s\}$ when $s \geq \frac{1+\sqrt{5}}{2}$. □

3.1 The range $1 \leq s < (1+\sqrt{17})/4$

Algorithm H':

If $p_i + L_s^i \leq s(1+\frac{1}{2s+1})$ **then** assign p_i to P_s **else** assign p_i to P_1

Theorem 9 *Algorithm* H' *is* $(1+\frac{1}{2s+1}) = \frac{2+2s}{2s+1}$ *-competitive for all* s *in the interval* $1 \leq s \leq \frac{1+\sqrt{17}}{4} \cong 1.2808$.

Proof. Let us fix the constant $c = \frac{1}{2s+1}$. We first observe that $s \in [1, \frac{1+\sqrt{17}}{4}]$ implies

$$1 + c = \frac{2+2s}{2s+1} \geq s. \tag{1}$$

Let us denote with p_k the first task such that $p_k + L_s^k > s(1+c)$.

If $L_s^k \geq s-c$, then the total size of p_k and all the successive tasks is less than $(1+s)-(s-c) = 1+c$. Thus, task p_k and all the others can be assigned to P_1 without violating the bound $1+c$.

If $L_s^k < s-c$, then the incoming task p_k is necessarily greater than $s(1+c)-(s-c) = c(s+1)$. Note that all the other tasks will be assigned to P_s. In fact, from $c = \frac{1}{2s+1}$ the inequality $(1+s)-c(s+1) \leq s(c+1)$ follows.

If $p_k \leq 1+c$, then the bound $H \leq (1+c)$ is guaranteed.

If $p_k > 1+c$, then from inequality (1) it follows that the off-line optimum is at least p_k/s, and so the ratio H/Z^* is not larger than $\frac{p_k}{p_k/s} = s \leq 1+c$. □

3.2 The range $(1+\sqrt{17})/4 \leq s < \sqrt{2}$

Algorithm H'':

If $p_i + L_s^i \leq s^2$ **then** assign p_i to P_s **else** assign p_i to P_1

Theorem 10 *Algorithm H'' is s-competitive for all $s \geq \frac{1+\sqrt{17}}{4}$.*

Proof. Let us consider the first task p_k such that $p_k + L_s^k > s^2$.

If $L_s^k > 1$, then the total size of p_k and all the successive tasks sum up to less than s. This means that no task can force the algorithm to load P_1 more than s and P_s more than s^2. Hence, the ratio H''/Z^* is not greater than s.

If $L_s^k < 1$, then $p_k > s^2 - 1$. Let us denote by R the sum of the successive tasks (p_k excluded). We know that $L_s^k + p_k + R = 1+s$, that is $L_s^k + R = 1+s-p_k < 2+s-s^2 \leq s^2$ (the latter inequality holds for $s \geq \frac{1+\sqrt{17}}{4}$. Thus, p_k will be assigned to P_1 and the load R will be assigned to P_s. The algorithm will produce a makespan $H'' = \max(p_k, s)$. If $p_k \leq 1$, then the ratio is $H''/Z^* = s$, otherwise the ratio is $H''/Z^* \leq \max(p_k, s)/p_k = \max(1, \frac{s}{p_k}) < s$. □

3.3 The range $\sqrt{2} \leq s$

Algorithm H''':

If $p_i + L_1^i \leq 1 + \frac{1}{s+1}$ **then** assign p_i to P_1 **else** assign p_i to P_s

Theorem 11 *Algorithm H''' is $(1+\frac{1}{s+1}) = \frac{s+2}{s+1}$-competitive for all $s \geq 1$.*

Proof. Let us denote by x the final load L_1 of processor P_1 in the heuristic solution. By the definition of the algorithm, we have $x \leq \frac{s+2}{s+1}$. If $\frac{1}{s+1} \leq x \leq 1$, then $H''' = (s+1-x)/s \leq (s+\frac{s}{s+1})/s = 1+\frac{1}{s+1}$; and if $x > 1$, then $H''' = x$. Thus, $H'''^* \leq \frac{s+2}{s+1}$ whenever $\frac{1}{s+1} \leq x \leq \frac{s+2}{s+1}$.

Suppose $x < \frac{1}{s+1}$, and denote $y = \frac{1}{s+1} - x$. Then each task assigned to P_s is larger than $1+y$. Hence, either $Z^* = H'''$ or Z^* is obtained by assigning the smallest task exceeding 1 to P_1 and all the other tasks to P_s. In the latter case we have $Z^* > 1+y$ for some value $0 < y \leq \frac{1}{s+1}$. Thus,

$$\frac{H'''}{Z^*} < \frac{s+1-x}{s(1+y)} = \frac{s+\frac{s}{s+1}+y}{s(1+y)} < 1 + \frac{1}{s+1}$$

since the worst upper bound for nonnegative y would occur with $y = 0$. □

References

[1] E. Angelelli (2000), Semi on-line scheduling on two parallel processors with known sum and lower bound on the size of the tasks, *Central European Journal of Operations Research* **8**, 285 – 295.

[2] E. Angelelli, M.G. Speranza, Zs. Tuza (2003), Semi on-line scheduling on two parallel processors with upper bound on the items, *Algorithmica* **37**, 243 – 262.

[3] E. Angelelli, M.G. Speranza, Zs. Tuza (2006), New bounds and algorithms for on-line scheduling: two identical processors, known sum and upper bound on the tasks, *Discrete Mathematics and Theoretical Computer Science* **8**, 1 – 6.

[4] Y. Cho, S. Shani (1980), Bounds for list schedules on uniform processors, *SIAM Journal of Computing* **9**, 91 – 103.

[5] L. Epstein, J. Noga, S. Seiden, J. Sgall, G. Woeginger (2001), Randomized on-line scheduling on two uniform machines, *Journal of Scheduling* **4**, 71 – 92.

[6] L. Epstein, L. M. Favrholdt (2002), Optimal non-preemptive semi-online scheduling on two related machines, *Journal of Algorithms* **57**, 49 – 73.

[7] L. Epstein, L. M. Favrholdt (2002), Optimal preemptive semi-online scheduling to minimize makespan on two related machines, *Operations Research Letters* **30**, 269 – 275.

[8] R.L. Graham (1966), Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* **45**, 1563 – 1581.

[9] Y. He, G. Zhang (1999), Semi on-line scheduling on two identical machines, *Computing* **62**, 179 – 187.

[10] H. Kellerer, V. Kotov, M.G. Speranza, Zs. Tuza (1997), Semi on-line algorithms for the partition problem, *Operations Research Letters* **21**, 235 – 242.

[11] K. Pruhs, J. Sgall, E. Torng (2004), Online scheduling, In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, ed. J. Y-T. Leung, chapter 15, pages 15-1 – 15-41, CRC Press.

[12] D. Sleator, R.E. Tarjan (1985), Amortized efficiency of list update and paging rules, *Communications of ACM* **28**, 202 – 208.

[13] J. Wen, D. Du (1998), Preemptive on-line scheduling for two uniform processors, *Operations Research Letters* **23**, 113 – 116.

Determining Rules in Fuzzy Multiple Heuristic Orderings for Constructing Examination Timetables

Hishammuddin Asmuni
Faculty of Computer Science & Information System, Universiti Teknologi Malaysia,
81310 Skudai, Johor, Malaysia, hishamudin@utm.my

Edmund K. Burke, Jonathan M. Garibaldi
School of Computer Science & IT, University of Nottingham, Nottingham, NG8 1BB, UK,
{ekb, jmg}@cs.nott.ac.uk

Barry McCollum
School of Computer Science, Queen's University Belfast, Belfast, BT7 1NN, UK, b.mccollum@qub.ac.uk

This paper presents alternative methods for producing fuzzy models for the *fuzzy multiple heuristic ordering* technique that we previously introduced for the construction of examination timetables. The effects of altering the rules within the fuzzy inference system were investigated. Four alternative rule 'tuning' approaches are described in detail and their results are presented and compared.

Keywords: Timetabling, fuzzy sets, fuzzy model identification, fuzzy rule tuning

1 Introduction

In the process of examination timetable construction, the order in which exams are assigned to time slots has been shown to have a major effect on the eventual solution [6]. An assessment of how difficult it is to place a given exam into a timetable (in effect, some measure of how hard it is to satisfy the constraints relevant to the particular exam) is often used to guide the order of placement. The usual strategy is to place the most difficult exams first, on the basis that it is better to leave the easier exams until later in the process when there are fewer time slots remaining. There are many different criteria that may be used when assessing this difficulty.

A common approach has been to employ graph based heuristics (a heuristic is an approximate rule or a 'rule-of-thumb' [5]) to provide a quantitative indication of difficulty. This measure is then used to determine the order in which the exams are assigned into the timetable and, hence, are referred to as 'heuristic orderings'. Examples of such heuristics are the number of other exams in conflict with the given exam, the number of students enrolled on each exam, etc. For detailed descriptions of these heuristic orderings, please refer to Asmuni *et al.* [3]. In our previous papers [1, 3], we explored how fuzzy techniques [7] could be employed to combine multiple heuristics within the construction of examination timetables. Particularly, we proposed the use of fuzzy inference systems to combine multiple heuristics *simultaneously* in order to provide a measure of the difficulty of placing each exam. This measure was then used to order (rank) the exams for assignment. Various combinations of heuristics were investigated in the construction process. In order to investigate the wider applicability of this novel fuzzy approach, the techniques were also applied to the domain of course timetabling [2]. However, to date, we have only been concerned with tuning the membership functions of the fuzzy variables, utilising a set of pre-defined (and fixed) fuzzy rules. This paper presents a series of experiments which were undertaken to explore the influences of tuning the fuzzy rules in fuzzy inference systems for which the membership functions had been fixed.

2 Method

The following heuristic orderings have been widely used to determine the order of placement of examinations in *sequential construction* algorithms:

Largest Degree (*LD*): Exams are ranked in descending order by the number of exams in conflict — i.e. priority is given to exams with the greatest number of exams in conflict.

Largest Enrollment (*LE*): Exams are ranked in descending order by the number of students enrolled in each of the exams.

Least Saturation Degree (*SD*): Exams are ranked in increasing order by the number of valid time slots remaining in the timetable for each exam.

Largest Coloured Degree (*LCD*): This heuristic is based on *LD*. For this heuristic, only exams which have been already assigned to the schedule are considered to cause conflict.

Weighted Largest Degree (*WLD*): This heuristic is also based on *LD*. Besides the number of exams, the total number of students involved in the conflict is also taken into account.

The general framework for constructing timetables using fuzzy inference systems utilising various combinations of these three heuristics has previously been described in Asmuni *et al.* [3]. Space limitations preclude the inclusion of the details of the methodology here, but in essence a fuzzy inference system was created that took three single heuristics as inputs and produced an overall assessment of each exam's 'difficulty to be placed', termed the *examweight*, as the output. In this study, the effect of two further fuzzy model determination techniques were investigated. Firstly, a simple enumerative search was implemented for tuning the fuzzy rules for the fuzzy system utilising a combination of the first three heuristics above (termed the *Fuzzy LD+SD+LE Model*) for which the membership functions had been previously tuned (see Section 2.1). Secondly, a random search was implemented to create fuzzy systems utilising combinations of all five heuristics above, in which the fuzzy model parameters (specifying both membership functions and rules) were determined by random selection (see Section 2.2). This allowed for a wider exploration of the total search space of alternative fuzzy models (which is vast).

2.1 Tuning Fuzzy Rules with Fixed Membership Functions

The objective of these experiments was to investigate whether tuning the fuzzy rules would offer any improvement in performance over the previously tuned and then fixed set of fuzzy rules. For this purpose, the best membership functions identified in experiments reported in [1] were implemented for the respective data sets. As we used the fuzzy multiple ordering that considered three heuristics simultaneously (i.e. *Fuzzy LD+SD+LE Model*), the rules shown in Fig. 6 of [1] were used as the benchmark fuzzy rule-set. The original fuzzy rule-set is detailed in Table 1, where the number in the cell represents the rule number. As an example, rule 22 is read as:

IF *LD* is *high* AND *LE* is *small* AND *SD* is *medium* THEN *examweight* is *small*

In the tuning process, the only modification allowed was in the consequence part of each rule. Six possible values for the consequence part were defined: *not_in_use*, *very small*, *small*, *medium*, *high* and *very high*. If the *not_in_use* value was assigned to the consequence part of a rule, that means the rule was not applicable. For each rule, its consequence part was changed by assigning one of the six possible values in the sequence of *not_in_use*, *very small*, *small*, *medium*, *high* and *very high*, iteratively. Considering 27 fuzzy rules and six possible values that could be assigned to the consequence part of each rule, there were 162 possible sets of fuzzy rules. Although the initial number of rules was 27, the number of rules might be reduced if, by doing so, the solution quality improved. Each rule-set was tested over three runs of the sequential construction algorithm. Hence,

Table 1: Fuzzy Rule set for *Fuzzy LD+LE+SD Model*

LE	***LD***								
	S			*M*			*H*		
	SD			***SD***			***SD***		
	S	*M*	*H*	*S*	*M*	*H*	*S*	*M*	*H*
S	S^{1}	VS^{4}	VS^{7}	S^{10}	S^{13}	VS^{16}	M^{19}	S^{22}	S^{25}
M	S^{2}	S^{5}	VS^{8}	H^{11}	M^{14}	M^{17}	H^{20}	M^{23}	M^{26}
H	H^{3}	S^{6}	S^{9}	H^{12}	M^{15}	M^{18}	VH^{21}	H^{24}	M^{27}

VS=very small
S=small
M=medium
H=high
VH=very high

at the end of the experiment, for each data set, three timetable solutions were obtained for each of the 162 rule-sets. The fuzzy rules set with the lowest penalty cost were selected as the 'best' set of rules for that specific data set. Two experiments were conducted:

- *Tuned Fuzzy Rules 1*— The set of tuned fuzzy rules that gave the best current solution quality was kept and used as the initial set of rules for the next step of the tuning process. A simple deepest descent enumerative search algorithm was employed in this experiment.
- *Tuned Fuzzy Rules 2*— Each of the rules was changed in isolation, no changes made in the earlier iterations were taken into account. Hence, after each change to the consequence part of any rule, the rules were reinstated to the initial configuration as shown in Table 1, before moving to the next iteration (i.e. setting another value in a consequence part).

2.2 Randomly Generated Fuzzy Models

The aim of this experiment was to examine alternative approaches for determining fuzzy models and to explore a larger space of possible fuzzy models. Instead of using fuzzy models in which either the membership functions or the rules were fixed, a stochastic approach was utilised to define the fuzzy model. In order to make the resultant fuzzy systems more manageable, only combinations of three heuristics selected from the five specified above (*LD*, *LE*, *SD*, *LCD* and *WLD*) were generated (the rule-sets for fuzzy systems featuring five heuristics would be enormous).

In the implementation, the first step was to randomly select which three heuristic orderings would be considered simultaneously. The next step was to create a set of fuzzy rules for the selected heuristic orderings, which were also selected in random fashion. Any rule contains at least one antecedent, up to a maximum of three antecedents. The last step was to choose centre points (*cp*) for membership functions for all of the fuzzy variables. The *cp* parameter defines the right-hand point of the *small*, the centre-point of the *medium* and the left-hand point of the *high* membership functions, as illustrated in Fig. 1. As a fuzzy system with three input and one output variables was implemented, four *cp* points needed to be randomly chosen.

Integer values were used to encode the heuristic orderings and fuzzy rules, as shown in Table 2. An example is presented in Figure 2 to show how the random fuzzy model was created. In *STEP 1*,

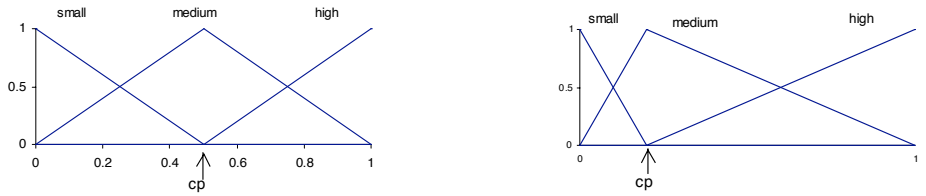


Figure 1: An illustration of the effect of the *cp* parameter on the membership functions of a variable

Table 2: Integer codes assigned to fuzzy model parameters

Heuristic	*LD*	*LE*	*SD*	*LCD*	*WLD*
Heuristic Code	1	2	3	4	5

Antecedent linguistic variable	not_inuse	small	medium	high
Antecedent Code	0	1	2	3

Consequence linguistic variable	not_inuse	very small	small	medium	high	very high
Consequence Code	0	1	2	3	4	5

the three heuristic orderings chosen are identified as *LE*, *SD* and *WLD*. Based on these heuristic orderings, the randomly generated rules are translated into '*IF ... THEN ...*' form. The rules are presented in a two dimensional array. Each row of the array represents one rule. In each row, the first column corresponds to the antecedent for the first heuristic ordering, the second column corresponds to the antecedent for the second heuristic and the third column corresponds to the third heuristic; the last column corresponds to the consequence part (i.e. *examweight*). In the example, three rules are randomly generated and their translated form are given. Notice that *Rule 2* only consists of two antecedents as *SD* is set to *not_in_use* (antecedent code = 0). This rule generation was performed without any concern as to the meaning of the rule — any rule was accepted even if it contradicted the 'common sense' of the relevant heuristic ordering. Next, in *STEP 3*, four *cp* points are randomly picked and a graphical representation of the membership functions that are generated is given. Again, the first three elements of the array correspond to the membership functions for the three chosen heuristic ordering in the sequence order; while the last element represents the *cp* point for *examweight*.

In order to evaluate this stochastic approach to fuzzy model determination, two experiments were performed as follows:

- *Random Model 1*: Experiments were performed for 100 iterations for each data set. In each iteration, a new fuzzy model was created by randomly choosing the heuristic orderings, 27 fuzzy rules and the four *cp* points for the membership functions. Each fuzzy model was tested three times within the sequential constructive algorithm
- *Random Model 2*: Experiments were conducted for 1000 iterations for nine of the data sets, while for *CAR-F-92*, *CAR-S-91* and *UTA-S-92*, the experiments were run for 100 iterations. For this experiment, the heuristic orderings and *cp* points were randomly choosen only once for each data set. Initially, the fuzzy rule-set was empty. In the first iteration, a fuzzy rule was randomly created and specified to be the first rule. Having created the fuzzy model, the sequential constructive algorithm was run three times. The best timetable constructed was set as a 'benchmark'. For each of the remaining iterations, a fuzzy rule was randomly created and appended to the list of rules. The sequential constructive algorithm was again run three times with the new fuzzy model (i.e. only the rules were changed). The rule was kept if a better solution was obtained with this new fuzzy model, and the new best solution was recorded as the new benchmark; otherwise, the newly added rule was removed. This process continued until the number of iterations exceeded the maximum number of iterations allowed for the particular data set.

In both experiments, non-applicable rules (rule in which the consequence part was *not_in_use* or rules in which all the antecedents were *not_in_use*) were removed. As the fuzzy rules were randomly

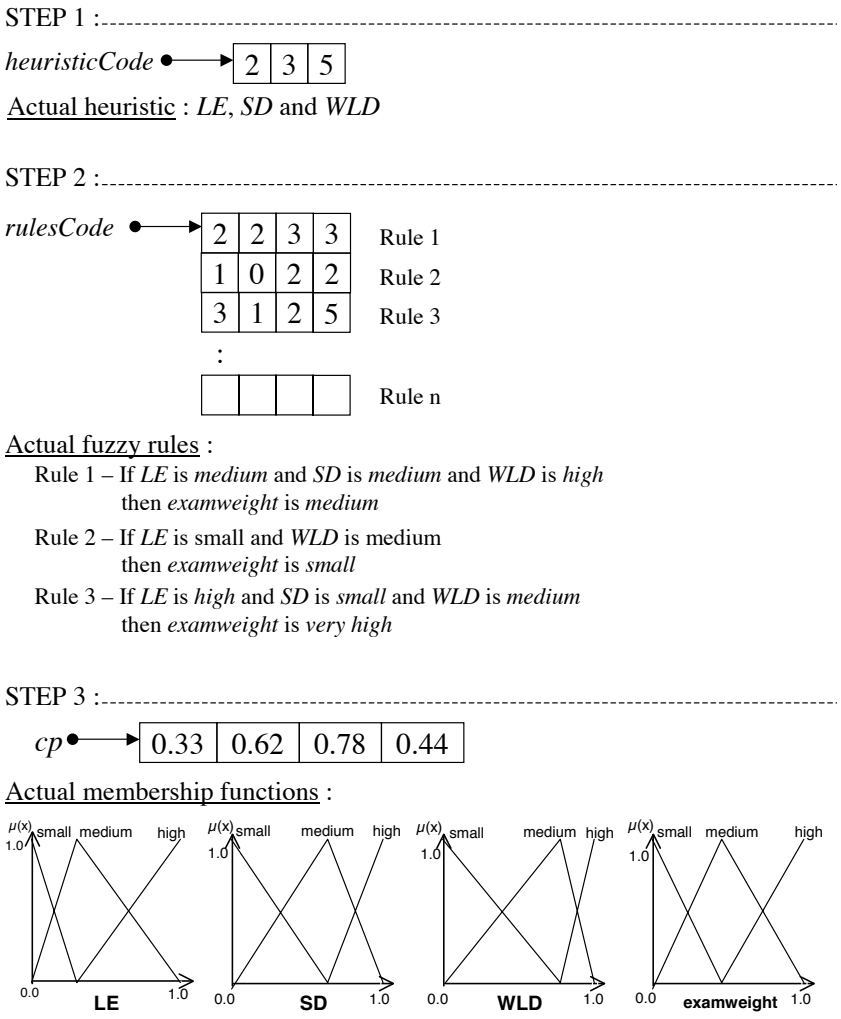


Figure 2: An example of defining a random fuzzy model

selected, in the case of experiment with *Random Model 1*, it was possible to have a fuzzy model that contained less than 27 rules. Note that the choice of number of repeats (three constructions and 100 / 1000 model iterations) was made arbitrarily on the basis of keeping the total computational time manageable.

3 Implementation

3.1 Problem Description

The experiments were carried out with 12 benchmark data sets made publicly available by Carter *et al.* [6]. Table 3 reproduces the problem characteristics. A proximity cost function was used to measure the timetable quality. This research deals with the uncapacitated version of the problem — i.e. the maximum room capacity for each time slot was not taken into account. Only feasible timetables were accepted. The penalty function was taken from Carter *et al.* [6]. It is motivated by the goal of spreading out each student's examination schedule. If two exams scheduled for a particular student are t time slots apart, the weight is set to $w_t = 2^{5-t}$ where $t \in \{1, 2, 3, 4, 5\}$.

Table 3: Examination Timetabling Problem Characteristics

Data Set	Number of time slots	Number of exams	Number of students	Conflict density
CAR-F-92	32	543	18419	0.14
CAR-S-91	35	682	16925	0.13
EAR-F-83	24	190	1125	0.27
HEC-S-92	18	81	2823	0.42
KFU-S-93	20	461	5349	0.06
LSE-F-91	18	381	2726	0.06
RYE-F-92	23	486	11483	0.08
STA-F-83	13	139	611	0.14
TRE-S-92	23	261	4360	0.18
UTA-S-92	35	622	21266	0.13
UTE-S-92	10	184	2750	0.08
YOR-F-83	21	181	941	0.29

The weight is multiplied by the number of students that sit for both of the scheduled exams. The average penalty per student is calculated by dividing the total penalty by the total number of students. The following formulation was used (adapted from Burke *et al.* [4]), in which the goal is to minimize:

$$\frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} s_{ij} w_{|p_j - p_i|}}{S},$$

where N is the number of exams, s_{ij} is the number of students enrolled in both exam i and j, p_i is the time slot where exam i is scheduled, and S is the total number of students; subject to $1 \leq |p_j - p_i| \leq 5$.

3.2 Results

Table 4 shows a comparison of the results obtained using fixed and tuned fuzzy rules. The first column indicates the penalty cost for the timetable solution of each data set that has been constructed with a set of fixed fuzzy rules (extracted from the ninth column of Table 2 in [1]). In the next two columns, the qualities of the timetable solutions produced using the sequential constructive algorithm with *Tuned Fuzzy Rules 1* and *Tuned Fuzzy Rules 2* approaches are given. In the fifth and sixth columns, the qualities of the timetable solutions produced by the experiments outlined in Section 2.2 are given. In Table 4, the best results across all experiments for each data set is highlighted in bold font. It can be seen that, in all data sets, better solutions were produced by tuning the fuzzy rules (either by *Tuned Fuzzy Rules 1* or *Tuned Fuzzy Rules 2*), compared to the approach that only used fixed fuzzy rules (column one), the only exception being *KFU-S-93* for which no improvement was observed. These results show that tuning the fuzzy rules produced considerably better timetable solutions. Although the results only show small improvements (in the range of 0.01 to 1.43), this evidence indicates that performing fuzzy rule tuning does give a considerable performance advantage.

In [1], we demonstrated that combinations of three heuristic ordering generally produced better solutions compared to combinations of two heuristic orderings. However, in two cases (*CAR-F-92* and *EAR-F-83*), it was observed that two heuristic orderings outperformed three heuristic orderings. Following the fuzzy rule tuning here, the *EAR-F-83* data set now has a penalty cost equal to 36.16. This penalty cost value is smaller than the penalty cost incurred when the *Fuzzy SD+LE* was used — i.e. 36.99 (see Table 2 in [1]). Although the result produced by the *Fuzzy SD+LE* model for the *CAR-F-92* (see Table 2 in [1]) still outperformed the result obtained in this experi-

Table 4: A comparison of results

Data Set	Fixed Fuzzy Rules (from [1])	*Tuned Fuzzy Rules 1*	*Tuned Fuzzy Rules 2*	*Random Model 1*	*Random Model 2*
CAR-F-92	4.52	4.51	4.51	4.59	**4.32**
CAR-S-91	5.24	**5.19**	**5.19**	5.58	5.54
EAR-F-83	37.11	**36.16**	36.64	40.93	37.05
HEC-S-92	11.71	11.61	**11.60**	12.55	12.31
KFU-S-93	15.34	15.34	15.34	15.74	**15.03**
LSE-F-91	11.43	**11.35**	**11.35**	12.58	12.65
RYE-F-92	10.30	10.02	10.05	10.58	**9.75**
STA-F-83	159.15	159.09	160.79	159.22	**158.64**
TRE-S-92	8.64	8.62	**8.47**	9.24	8.79
UTA-S-92	3.55	**3.52**	**3.52**	3.69	4.31
UTE-S-92	27.64	27.64	**27.55**	29.77	29.10
YOR-F-83	40.68	**39.25**	39.79	43.88	42.30

ment, overall the results indicate the potential of expanding the tuning of the fuzzy model to also incorporate tuning the fuzzy rules.

If we now compare the results obtained for rule tuning (the third and fourth columns) to the results obtained by random model generation (the fifth and sixth columns) it can be seen that the best result for eight of the data sets was obtained by fuzzy models that were developed using tuned membership functions with tuned fuzzy rules. Although experiments which applied *Random Model 1* did not produce any best results, the experiments that used *Random Model 2* actually produced four best results. These best results were obtained using the following fuzzy models:

CAR-F-92:

- heuristic orderings: *LCD*, *LE* and *SD*
- *cp* points for membership functions: 0.550, 0.110, 0.296, and 0.132
- number of fuzzy rules: 16

KFU-S-93:

- heuristic orderings: *WLD*, *SD* and *LE*
- *cp* points for membership functions: 0.021, 0.721, 0.351, and 0.095
- number of fuzzy rules: 48

RYE-F-92:

- heuristic orderings: *LE*, *WLD* and *LCD*
- *cp* points for membership functions: 0.679, 0.358, 0.001, and 0.708
- number of fuzzy rules: 9

STA-F-83:

- heuristic orderings: *WLD*, *SD* and *LE*
- *cp* points for membership functions: 0.309, 0.739, 0.408, and 0.595
- number of fuzzy rules: 17

One possible reason why only four best results were found is the fact that the number of iterations in the experiments (100 for *Random Model 1* and 1000 for *Random Model 2*) was quite small when compared to the huge search space that needs to be explored in order to find the 'optimal' fuzzy model. Taking a different view, one should notice that performing tuning of membership functions and fuzzy rules in separate stages is better than performing both membership functions and fuzzy rules tuning at the same stage (as in the *Random Model 2* approach). It also worthy of

mention that only 16, 9 and 17 fuzzy rules are required to produced the best solutions obtained for *CAR-F-92*, *RYE-F-92* and *STA-F-83* respectively. This indicates that not all possible rules are require to be utilised in such a fuzzy system in order to get a good solution. Fewer rules makes the fuzzy model more understandable for the developer and user. Therefore, a more sophisticated optimisation approach could almost certainly be devised to tackle the tuning process more systematically.

4 Conclusion

The overall aim of this paper was to investigate the effect of altering the fuzzy rules for fuzzy multiple heuristic ordering in measuring the difficulty of assigning exams into time slots. Having implemented two alternative approaches for altering the fuzzy rules (one for tuning the rules once membership functions have been tuned and fixed and the other for randomly generating alternative rule-sets), the results obtained demonstrated that better solutions can be generated. Unfortunately the space of all possible fuzzy models is vast and there remains much scope for the investigation of more sophisticated methods to efficiently search this vast space in order to find effective models. Ultimately, for the method to be genuinely applicable in the real world, it would be necessary to determine a single generically applicable fuzzy model. To what extent this is possible is currently an open research issue.

Acknowledgements. This research work is supported by the Universiti Teknologi Malaysia (UTM) and the Ministry of Science, Technology and Innovation (MOSTI) Malaysia.

References

[1] H. Asmuni, E. K. Burke, and J. M. Garibaldi. A Comparison of Fuzzy and Non-Fuzzy Ordering Heuristics for Examination Timetabling (2004), A. Lotfi, editor, *P*roceedings of 5th International Conference on Recent Advances in Soft Computing 2004, 288 – 293.

[2] H. Asmuni, E.K. Burke, and J.M. Garibaldi (2005), Fuzzy Multiple Heuristic Orderings for Course Timetabling Problem, B. Mirkin and G. Magoulas, editors, *P*roceedings of the 2005 UK Workshop on Computational Intelligence (UKCI2005), London, UK, September 2005, Birkbeck University of London, 302 – 309.

[3] H. Asmuni, E.K. Burke, J.M. Garibaldi, and Barry McCollum (2004), Fuzzy Multiple Heurisitc Orderings for Examination Timetabling, E. K. Burke and M. Trick, editors, *P*ractice and Theory of Automated Timetabling V (PATAT 2004, Pittsburg USA, August 2004, Selected Revised Papers), *L*ecture Notes in Computer Science **3616**, Berlin, 2005. Springer, 334 – 353.

[4] E.K. Burke, Y. Bykov, J. Newall, and S. Petrovic (2004), A Time-Predefined Local Search Approach to Exam Timetabling Problems, *I*IE Transactions **36**(6), 509 – 528.

[5] E.K. Burke and G. Kendall editors (2005), *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*, Springer.

[6] M.W. Carter, G.G. Laporte, and S.Y. Lee (1996), Examination Timetabling: Algorithmic Strategies and Applications. *J*ournal of the Operational Research Society **47**, 373 – 383.

[7] L.A. Zadeh (1965), Fuzzy Sets. *I*nformation and Control **8**, 338 – 353.

A Simulated Annealing Hyper-heuristic: Adaptive Heuristic Selection for Different Vehicle Routing Problems

Ruibin Bai, Edmund K. Burke
School of Computer Science & IT, University of Nottingham, Nottingham NG8 1BB, UK, {rzb, ekb}@cs.nott.ac.uk
Michel Gendreau
CIRRELT, Université de Montréal, C.P. 6128, Succursale Centre-ville, Montréal, Canada H3C 3J7, michelg@crt.umontreal.ca
Graham Kendall
School of Computer Science & IT, University of Nottingham, Nottingham NG8 1BB, UK, gxk@cs.nott.ac.uk

1 Introduction

One of the motivations of hyper-heuristic research is to investigate the development of adaptive decision support systems that can be applied to a range of different problems and different problem instances [5]. One possible approach is to dynamically adjust the preferences of a set of simple low-level heuristics (or neighbourhood operators) during the search. Hyper-heuristics have been used to address a variety of search problems, including educational timetabling [6, 2, 8, 7], scheduling [10], and packing and layout optimisation [1, 9]. In this research, we intend to further investigate an adaptation mechanism in the context of vehicle routing problems (VRP).

VRP has numerous practical applications and it has been one of the most studied combinatorial optimisation problems in the past fifty years. Several variants of VRP have been addressed. Examples include capacitated VRP, VRP with time windows, multiple depot VRP, periodic VRP, stochastic VRP, and VRP with pickup and delivery. However, most existing algorithms for these problems are tailored for a particular problem variant and it is generally difficult to reuse the same algorithm for a different scenario without significant algorithm redevelopment or parameter-tuning. Recently, there are a few important research papers being carried out to improve the generality of algorithmic methods so that they can handle problems across more than one VRP variant [11, 12]. In both papers, the searches are based on combinations of "destruction-repair operators", guided by some problem dependent penalty functions. In this abstract, we want to continue this research direction using a simulated annealing hyper-heuristic technique in order to investigate how the algorithm can intelligently choose between different neighbourhood operators (heuristics) according to the different problems. In particular, we consider two of the most popular variants of vehicle routing problems (capacitated VRP and VRP with time windows) and investigate the adaptation of the heuristic-selection mechanism across these variants and at different stages of the search. Specifically, we want to investigate: 1). How the hyper-heuristic adapts to these two types of vehicle routing problems by changing preferences of low-level heuristics and whether the hyper-heuristic can automatically identify heuristics that are particularly good for a given type of vehicle routing problem? 2). How the hyper-heuristic adapts its selection decision during different stages of the search when solving a particular problem instance?

2 Vehicle Routing Problems

The Vehicle Routing Problem (VRP) [16] describes a whole class of problems in which a fleet of vehicles (based at one or several depots) must serve geographically dispersed customer demands.

The objective is usually to minimise the total cost, in terms of vehicle fleet size, travelling distance and/or travelling time. The VRP is a well known NP-Hard problem for which no polynomial time algorithm is known.

2.1 The capacitated vehicle routing problem

The capacitated vehicle routing problem (CVRP) is a version of VRP, in which a fleet of m vehicles of the same (limited) capacity Q is available and the objective is to minimise total travel distance. The CVRP can be formulated as a connected graph $G = (V, E)$ of $n + 1$ nodes, with each node $v_i \in V, i = 1, ..., n$ representing a customer of demand q_i and vertex v_0 representing the depot. The problem is thus to find a set of vehicle routes, denoted as $R = \{r_j | j = 1, ..., m\}$, so that every customer is served by exactly one vehicle-visit and the total customer demand on each route r_j does not exceed the vehicle capacity Q [15].

2.2 Vehicle routing with time windows

Vehicle routing with time windows (VRPTW) is an important variant of VRP with many applications [3, 4]. It is very similar to CVRP but, in VRPTW, each customer is associated with a time window, which defines an interval wherein the customer has to be served. The interval at the depot is called the scheduling horizon. If a vehicle arrives at a customer before the time window of that customer, the service cannot start until time reaches the time window.

Note that even VRPTW itself has several variants. In this research, we are concerned with the same model that has been addressed by most researchers. That is, the graph is complete and the edge costs are symmetric. The objective is hierarchical: one first seeks to minimise the size of the fleet required to service customers and, in the second step, the total distance traveled by the fleet. Considering the hyper-heuristic approach used in this research, the solution can be represented by m linked lists, with each linked list $j \in m$ representing one vehicle schedule and each node in j representing a customer to be served by j in the same sequence.

3 Simulated annealing hyper-heuristics

We focus on a recently proposed simulated annealing hyper-heuristic framework [2] which has demonstrated generality across the university course timetabling problem, the nurse rostering problem, and the bin packing problem. The set of low-level heuristics (neighbourhood operators) for both the CVRP and VRPTW is drawn from the literature. These heuristics are briefly described below.

- **Relocate**. This heuristic moves a random customer from its current route to a different one.
- **2-opt* exchange**. This heuristic is specially designed for VRPTW. It is a generalisation of the 2-opt operator for TSP in order to satisfy the time-window constraints. The main modification over 2-opt is that the exchange maintains the direction of the original routes by introducing the customers with late time windows after customers with early time windows [13].
- **k-opt**. This is a very efficient neighbourhood operator for the travelling salesman problem.
- **Or-opt exchange**. This heuristic generally operates on a single route and changes the position of a random sub-route (no more than three customers) within the current route [13].

- **CROSS exchange**. This heuristic is a very effective heuristic for VRPTW. It exchanges two route-segments from two different routes and also preserves the orientation of customers [14].
- **Ejection-Chain**. This heuristic repeatedly moves conflicting customers, triggered by a random customer relocation operation, from their current routes to other ones until the feasibility of the solution is recovered [3].

4 Summary

In summary, we plan to utilise a simulated annealing based hyper-heuristic for two classes of vehicle routing problems (capacitated VRP and VRP with time windows). Our hypothesis is that the hyper-heuristic algorithm will produce good quality solutions across a range of problem instances, without having to resort to tuning parameters for each instance. We will report on the results of these experiments at the conference.

References

[1] R. Bai and G. Kendall (2005), An investigation of automated planograms using a simulated annealing based hyper-heuristics, *in* T. Ibaraki, K. Nonobe and M. Yagiura (eds), *Metaheuristics: Progress as Real Problem Solver*, Operations Research/Computer Science Interface Series **32**, Springer, 87 – 108.

[2] R. Bai, J. Blazewicz, E.K. Burke, G. Kendall and B. McCollum (2007), A simulated annealing hyper-heuristic methodology for flexible decision support, *Technical report*, School of CSiT, University of Nottingham, UK.

[3] O. Bräysy and M. Gendreau (2005), Vehicle routing problem with time windows, part I: Route construction and local search algorithms, *Transportation Science* **39**(1), 104 – 118.

[4] O. Bräysy and M. Gendreau (2005), Vehicle routing problem with time windows, part II: Metaheuristics, *Transportation Science* **39**(1), 119 – 139.

[5] E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross and S. Schulenburg (2003), Hyper-heuristics: An emerging direction in modern search technology, F. Glover and G. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer, 457 – 474.

[6] E.K. Burke, G. Kendall and E. Soubeiga (2003), A tabu-search hyperheuristic for timetabling and rostering, *Journal of Heuristics* **9**(6), 451 – 470.

[7] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic and R. Qu (2007), A graph-based hyper heuristic for educational timetabling problems, *European Journal of Operational Research* **176**(1), 177 – 192.

[8] E.K. Burke, S. Petrovic and R. Qu (2006), Case based heuristic selection for timetabling problems, *Journal of Scheduling* **9**(2), 115 – 132.

[9] K.A. Dowsland, E. Soubeiga and E.K. Burke (2007), A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation, *European Journal of Operational Research* **179**(3), 759 – 774.

[10] E. Hart, P. Ross and J.A. Nelson (1998), Solving a real-world problem using an evolving heuristically driven schedule builder, *Evolutionary Computing* **6**(1), 61 – 80.

[11] D.I. Mester and O. Bräysy (2005), Active guided evolution strategies for large-scale vehicle routing problems with time windows, *Computers & Operations Research* **32**, 1593 – 1614.

[12] D. Pisinger and S. Ropke (2007), A general heuristic for vehicle routing problems, *Computers & Operations Research* **34**, 2403 – 2435.

[13] J.-Y Potvin, T. Kervahut, B.-L. Garcia and J.-M. Rousseau (1996), The vehicle routing problem with time windows – part I: Tabu search, *INFORMS Journal on Computing* **8**, 158 – 164.

[14] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J.-Y. Potvin (1997), A tabu search heuristic for the vehicle routing problem with soft time windows, *Transportation Science* **31**(2), 170 – 186.

[15] P. Toth and D. Vigo (2002), Models, relaxations and exact approaches for the capacitated vehicle routing problem, *Discrete Applied Mathematics* **123**(1-3), 487 – 512.

[16] P. Toth and D. Vigo (eds): 2001, *The Vehicle Routing Problem*, Society for Industrial & Applied Mathematics. Philadelphia.

Sequencing a Single Machine with Due Dates and Deadlines: an ILP-based Approach to Solve Very Large Instances

Philippe Baptiste
CNRS LIX, Ecole Polytechnique, France, philippe.baptiste@polytechnique.fr
Federico Della Croce, Andrea Grosso
D.A.I., Politecnico di Torino, Italy, {federico.dellacroce, grosso}@polito.it
Vincent T'kindt
Laboratoire d'Informatique, Université de Tours, France, tkindt@univ-tours

We consider the problem of minimizing the weighted number of tardy jobs on a single machine where each job is also subject to a deadline that cannot be violated. We propose an exact method based on a compact integer linear programming formulation of the problem and an effective reduction procedure, that allows to solve to optimality instances with up to 30,000 jobs in size.

Keywords: Machine Scheduling, Algorithmics.

1 Introduction

This paper deals with the problem of minimizing the weighted number of tardy jobs on a single machine when both due dates and deadlines are specified for jobs. In the classical three-fields notation the problem is denoted $1|\bar{d}_i|\sum w_iU_i$: a job set $N = \{1, 2, \ldots, n\}$ is given, with weights, processing times, due dates and deadlines w_i, p_i, d_i and $\bar{d}_i$ respectively, for all $i \in N$. The goal is to find a sequence σ^* such that $C_i \leq \bar{d}_i$ for all the job completion times C_i, $i \in N$, and $F(\sigma^*) = \sum\{w_i \colon C_i > d_i\}$ is minimum — or equivalently, $f(\sigma^*) = \sum\{w_i \colon C_i \leq d_i\}$ is maximum. The jobs are executed without idle time or preemption, and all of them are available from time 0 on.

From a complexity point of view, the $1|\bar{d}_i|\sum w_iU_i$ problem is known to be NP-hard even if $w_i = 1$, for all $i \in N$ [1], or if there are no deadlines [2], but it is still unclear whether it is NP-hard in the strong or in the ordinary sense. On the other hand, the basic $1||\sum U_i$ problem is well known to be polynomially solvable by Moore's algorithm [3]. If release dates are present, the $1|r_i|\sum U_i$ problem is already NP-hard in the strong sense [4]. Very little work has appeared in literature for the $1|\bar{d}_i|\sum w_iU_i$ problem, a state-of-the-art exact algorithm being a branch and bound presented in [6] able to solve instances with up to 300 jobs.

A richer literature is available for the deadline-free variant: Potts and Van Wassenhove [5] gave a branch and bound algorithm for solving instances with up to 1,000 jobs, while M'Hallah and Bulfin [8] propose an exact algorithm capable of handling instances with up to 2,500 jobs. Several papers are also available for the variants with release dates (see [9, 10, 11]). However, the presence of release dates completely changes the structure of the problem.

We note that most of the works on this scheduling problem explicitly avoid to manage the LP relaxation by standard LP tools: in [5] an ILP model is presented, but a specific dynamic programming algorithm is used for solving the relaxation; in [6] no LP is used, the bound being obtained by dynamic programming and state-space relaxation; even in more recent papers like [8] a lagrangian approach is used, and the simplex method is avoided.

In this paper we present a compact integer linear programming formulation and, explicitly relying on an LP/ILP solver with minimum additional procedures, we design a simple exact algorithm

that is able to solve to optimality all instances with up to 30,000 jobs (within 275 seconds on the average) for the $1|\bar{d}_i|\sum w_iU_i$ problem and up to 50,000 jobs (within 316 seconds on the average) for the special deadline-free case $1||\sum w_iU_i$.

Throughout this Conference paper, we omit the proofs for conciseness.

2 ILP model and problem reduction

2.1 Basic properties and model

We first recall that a feasible solution for $1|\bar{d}_i|\sum w_iU_i$ is immediately defined by selecting an *early set* of jobs $E \subseteq N$ required to be early: each job $i \in N$, is then required to be completed within a maximum completion time

$$D_i = \begin{cases} d_i & \text{if } i \in E, \\ \bar{d}_i & \text{if } i \in N \setminus E. \end{cases}$$

A feasible sequence with early set E is a sequence where $C_i \leq D_i$ for all $i \in N$. We recall that such feasible sequences exist iff the particular sequence where the jobs appear in nondecreasing order of D_i is feasible.

Define $B_t = \{i \in N \colon \bar{d}_i \leq t\}$, $A_t = \{i \in N \colon d_i > t\}$, and let $T = (t_1, t_2, \ldots, t_m)$ be the nondecreasing sequence of the relevant time points in the problem with $t \in T$ iff $t = d_i$ or $t = \bar{d}_i$ for some $i \in N$. The optimal set of early jobs can be obtained by solving the following ILP model. Define binary variables x_i, $i \in N$, such that $x_i = 1$ iff job i is early.

$$\text{maximize } z = \sum_{i \in N} w_i x_i \tag{1}$$

subject to

$$\sum_{i \in B_t} p_i + \sum_{i \in N \setminus (B_t \cup A_t)} p_i x_i \leq t \qquad t \in T \tag{2}$$

$$x_i \in \{0, 1\}, \quad i \in N \tag{3}$$

With modern ILP solvers and hardware, model (1)–(3) is able to handle fairly large instances: in our experiments, all the considered examples with up to 4,000 jobs within reasonable average CPU times, although the time exceeded 1,000 seconds for the hardest instances. The failures for higher sizes were caused essentially by lack of memory — note that the number of nonzeros in the constraints (2) exhibits a $\mathcal{O}(n^2)$ growth in the worst case.

In our experience the solver can largely benefit from a problem preprocessing that is able to significantly reduce the number of jobs. Suppose a given job $j \in N$ is known to be early or tardy in an optimal solution: this defines its $D_j = d_j$ or $\bar{d}_j$. We define a *reduced* problem formulated on the job set $N' = N \setminus \{j\}$, with modified data

$$p'_i \equiv p_i, \; w'_i \equiv w_i \qquad i \in N', \tag{4}$$

$$d'_i = \begin{cases} \min\{d_i, D_j - p_j\} & \text{if } d_i \leq D_j \\ d_i - p_j & \text{if } d_i > D_j \end{cases} \qquad i \in N', \tag{5}$$

$$\bar{d}'_i = \begin{cases} \min\{\bar{d}_i, D_j - p_j\} & \text{if } \bar{d}_i \leq D_j \\ \bar{d}_i - p_j & \text{if } \bar{d}_i > D_j \end{cases} \qquad i \in N'. \tag{6}$$

The following result generalizes the *reduction theorem* presented in [5] for the deadline-free special case.

Property 1 *There exists a feasible sequence with early set E iff there exists a feasible sequence with early set $E' = E \setminus \{j\}$ for the reduced problem.*

Property 1 allows to remove job j from consideration and solve the reduced problem only, without loss of optimality.

We iteratively identify early or tardy jobs by variable fixing techniques, removing them from the problem by means of Property 1. Components needed for such reduction procedure are a quick method for solving the LP relaxation of (1)–(3) and a heuristic solution.

2.2 Solving the LP relaxation

For the LP relaxation, instead of using the dense formulation (1)–(3) we introduce a maximum profit flow problem. Define a graph $G(N \cup T, A)$ where nodes represent jobs and time points, whereas the arc set A is defined as

$$A = A_d \cup A_{\bar{d}} \cup A_T,$$

where

$$\begin{aligned} A_d &= \{(i, t_k) \colon i \in N, t_k \in T, t_k = d_i\}, \\ A_{\bar{d}} &= \{(i, t_k) \colon i \in N, t_k \in T, t_k = \bar{d}_i\}, \\ A_T &= \{(t_k, t_{k+1}) \colon t_k, t_{k+1} \in T, k = 1, \ldots, m-1\}. \end{aligned}$$

We formulate on G the following flow problem.

$$\text{maximize } z = \sum_{(i,d_i) \in A_d} \frac{w_i}{p_i} y_{i,d_i} \tag{7}$$

subject to

$$y_{i,d_i} + y_{i,\bar{d}_i} = p_i \qquad i \in N \tag{8}$$

$$y_{t_1,t_2} - \sum_{(i,t_1) \in A_d \cup A_{\bar{d}}} y_{i,t_1} = 0 \tag{9}$$

$$y_{t_k,t_{k+1}} - y_{t_{k-1},t_k} - \sum_{(i,t_k) \in A_d \cup A_{\bar{d}}} y_{i,t_k} = 0 \qquad k = 2, \ldots, m-1 \tag{10}$$

$$- y_{t_{m-1},t_m} - \sum_{(i,t_m) \in A_d \cup A_{\bar{d}}} y_{i,t_m} = -\sum_{i \in N} p_i \tag{11}$$

$$y_{t_k,t_{k+1}} \le t_k \qquad (t_k, t_{k+1}) \in A_T \tag{12}$$

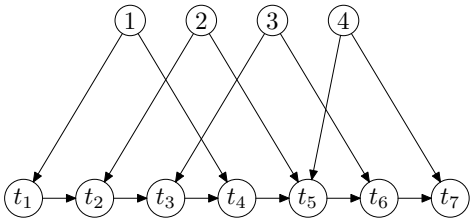
$$y_{ij} \ge 0, \ (i,j) \in A, \quad k = 1, \ldots, m-1. \tag{13}$$

The shape of G is sketched in Figure 1. Constraints (8)–(11) are flow balance constraints and constraints (12) are capacity constraints. Nodes $i \in N$ are sources injecting p_i units of flow each in the network, node t_m is the unique sink of the network, and the problem is balanced. All arcs have zero cost except the arcs $(i, d_i) \in A_d$ having cost $\frac{w_i}{p_i}$. All arcs have infinite capacity except the arcs $(t_k, t_{k+1}) \in A_T$ that have finite capacities t_k.

Property 2 *$(y_{ij} \colon (i,j) \in A)$ is a feasible flow of value z for (7)–(13) iff*

$$x_i = \frac{y_{i,d_i}}{p_i} \qquad (i \in N)$$

is a feasible solution of value z for (1)–(3).



$T = (t_1, \ldots, t_7) =$
$(d_1, d_2, d_3, \bar{d}_1, \bar{d}_2 = d_4, \bar{d}_3, \bar{d}_4)$
Node balances b_i's are as follows:
$b_1 = p_1,$
$b_2 = p_2$
$b_3 = p_3$
$b_4 = p_4$
$b_{t_1}, \ldots, b_{t_6} = 0$
$b_{t_7} = -(p_1 + p_2 + p_3 + p_4)$

Figure 1: Graph $G(N \cup T, A)$ for a four-jobs example.

Property 2 establishes a one-to-one correspondence between feasible solutions of the LP relaxation of (1)–(3) and (7)–(13). In our experiments we kept solving the flow model (7)–(13) that requires $\mathcal{O}(n)$ (instead of quadratic) space, thus overcoming the memory problems experienced with formulation (1)–(3) on large instances.

2.3 Heuristic solution and core problem

Once the LP relaxation is solved, we need to determine a heuristic solution of (1)–(3). Preliminary testing with a constructive procedure (first an initial solution having as early all the jobs corresponding to variables having value 1 in the lower bound solution is generated and then all the other jobs are tested one at a time for inclusion in the early set according to a greedy rule) did not reach satisfactory results. Notice that in all tests most of the variables present an integer value in the LP relaxation solution. Based on this consideration, we propose here a heuristic solution corresponding to the optimal solution of a *core problem*.

To this extent we introduce the following dominance property (easily provable by interchange argument).

Property 3 *Let $p_i \leq p_j, d_i \geq d_j, \bar{d}_i \leq \bar{d}_j$ and $w_i \geq w_j$ with at least one strict inequality. Then:*

- *if i is tardy also j must be tardy*
- *if j is early also i must be early.*

Let $X^* = [x_1^*, \ldots, x_n^*]$ be the optimal solution of the linear programming continuous relaxation of the problem. The set C of variables (jobs) in the core problem is determined by selecting

- all variables presenting non integer value in X^*,
- all variables x_j set to 0 in X^* and that are non dominated according to Property 3 by any other variable x_i set to 0 in X^*,
- all variables x_j set to 1 in X^* and that are dominated according to Property 3 by every other variable x_i set to 1 in X^*,

All the jobs $j \notin C$ are fixed early (if $x_j = 1$) or tardy (if $x_j = 0$) and removed via Property 1.

The core problem size has in all tests been always less than 5% of the original problem size and is therefore solvable to optimality in very short time by the ILP solver. In order to further improve the quality of the heuristic solution, we perform then a local search phase that uses the

optimal solution of the core problem as initial solution. Two feasible solutions $(\bar{x}_1, \ldots, \bar{x}_n)$ and $(\tilde{x}_1, \ldots, \tilde{x}_n)$ are neighbours if $\sum_{i \in N} |\bar{x}_i - \tilde{x}_i| = 2$, i.e. a job tardy and a job early are swapped. The corresponding neighbourhood can be generated and evaluated in $\mathcal{O}(n^3)$ time, and a best-improve exploration is adopted. Once again, to save time, a job j tardy (early) is swapped iff $\nexists i : x_i^* = 0$ ($x_i^* = 1$) dominating j (dominated by j) according to Property 3. In this way, in practice, always much less than 1% of the neighborhood is explored. Let denote by $\bar{z}$ the solution value provided by the local search phase.

2.4 Fixing jobs

Job fixing is performed by applying variable-fixing techniques from Integer Linear Programming. Given an optimal basis B^* for (1)–(3), let

$$\begin{aligned} z =& z^* + \sum_{x_i \notin B^*} \bar{c}_i x_i \\ x_j =& x_j^* + \sum_{x_i \notin B^*} \alpha_{ji} x_i \qquad (x_j \in B^*) \\ & x_i \geq 0,\ i \in N \end{aligned}$$

be the corresponding reformulation with $\bar{c}_i$ being the reduced cost of variable x_i and α_{ji} being the updated coefficient of variable x_i in the constraint related to the in-base variable x_j. We apply the following fixing rules.

For nonbasic variables:

(R1) fix $x_i = 0$ (job i tardy) if $\bar{c}_i < 0$ and $z^* + \bar{c}_i \leq \bar{z}$,

(R2) fix $x_i = 1$ (job i early) if $\bar{c}_i > 0$ and $z^* - \bar{c}_i \leq \bar{z}$.

For basic variables $x_j \in B^*$, we compute the branching penalties u_j, l_j for branching at $x_j = 1$ and at $x_j = 0$ respectively (often indicated as *pseudo-costs* — see, for instance, [12]):

$$u_j = (x_j^* - 1) \min \left\{ -\frac{\bar{c}_i}{\alpha_{ji}} : x_i \notin B^*, \bar{c}_i \alpha_{ji} \leq 0, \alpha_{ji} \neq 0 \right\}; \tag{14}$$

$$l_j = -x_j^* \min \left\{ \frac{\bar{c}_i}{\alpha_{ji}} : x_i \notin B^*, \bar{c}_i \alpha_{ji} \geq 0, \alpha_{ji} \neq 0 \right\}; \tag{15}$$

then we apply

(R3) fix $x_j = 0$ if $z^* + u_j \leq \bar{z}$,

(R4) fix $x_j = 1$ if $z^* + l_j \leq \bar{z}$.

Although sometimes overlooked in textbooks, this technique is known in integer programming as well as in constraint programming (see [7, 13] for an application to scheduling problems). Also, note that u_j, l_j are easily computed on the maximum profit flow problem — computing penalties for setting $y_{j,d_j} = p_j$, $y_{j,d_j} = 0$.

The complete reduction procedure works as follows — fixed jobs in steps (2) and (3) below are removed by means of Property 1.

1. the LP relaxation is solved via model (7)–(13),
2. jobs are fixed if possible, by rules (R1) and (R2),

3. jobs are fixed if possible, by rules (R3) and (R4),

4. if (R3), (R4) were successful in fixing jobs, steps (1)–(3) are reiterated on the reduced problem, else the procedure stops.

3 An exact algorithm

The solution algorithm we considered is a depth-first branch and bound procedure relying on problem reduction and model (1)–(3). We observed that the ILP model already reaches good performances for fairly large n: the limit seems to be memory consumption, due to the quadratic growing of the constraint matrix size as n increases. We then implemented a depth-first enumeration scheme which incorporates the reduction procedure sketched in Section 2.4: at each node, the LP relaxation of model (1)–(3) is solved via the equivalent network flow model, and the reduction procedure is applied. If the reduced problem allows to build an instance of (1)–(3) with no more than $1.4 \cdot 10^7$ nonzeros, such integer program is solved directly by calling the ILP solver. Otherwise the fractional variable x_i with largest max-min pseudo-cost, namely the one with $\max_{i:0<x_i^*<1}\{\min\{|l_i|, |u_i|\}\}$ value, is selected and binary branching is performed by setting $x_i = 0$ (job i tardy) and $x_i = 1$ (job i early) in the descendant nodes. The resulting algorithm is quite simple, and adds a minimal machinery on top of the ILP solver. The performances of the algorithm are further enhanced by incorporating Property 3 in the branching phase.

4 Computational results

4.1 Test instances

Following [6], we considered ten classes of instances structured as follows.

- The values for processing times $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$ are integers drawn randomly from the uniform distribution $[1, 100]$. We note that in [6] the range was $[1, 10]$.

- The due dates $d_1, \ldots, d_n$ are integers drawn randomly from the uniform distribution $[Pu, Pv]$, with $P = \sum_{i \in N} p_i$. Ten pairs of values for u, v were considered: $u \in \{0.1, 0.3, 0.5, 0.7\}$, $v \in \{0.3, 0.5, 0.7, 0.9\}$, $u < v$.

- The deadlines $\bar{d}_1, \ldots, \bar{d}_n$ are integers drawn for each job i from the uniform distribution $[d_i, 1.1P]$.

We generated 20 examples for each u, v class — thus creating batches of 200 examples — with size n ranging from 1000 to $15,000$. All the testing ran on a Pentium IV PC with 3 GHz clock and 1GB memory. The ILP solver used is XPRESS-MP by Dash Optimization.

4.2 Results

As previously remarked, model (1)–(3) already reaches fairly good performances for $n \leq 4000$ — see Table 1.

The solver did not manage to solve the whole $n = 5000$ batch because of memory problems. The enumerative algorithm was able to solve all the batches up to $n = 30000$, the worst case being a $25,000$- job instance with distribution $u = 0.1$, $v = 0.5$ that required 3982 seconds. The results are summarized in Table 2.

	CPU Time (s)	
n	avg	max
1000	7.2	21.3
2000	44.8	170.9
3000	135.3	577.2
4000	289.7	1278.5

Table 1: Performances of model (1)–(3).

	CPU Time (s)		NODES		UB-GAP$_{\%}$		LB-GAP$_{\%}$	
n	avg	max	avg	max	avg	max	avg	max
1000	0.9	4.7	1.0	1	0.008	0.084	0.001	0.037
2000	2.2	22.6	1.0	1	0.003	0.041	0.001	0.042
3000	3.6	12.4	1.0	1	0.002	0.026	$< 10^{-3}$	0.016
4000	5.9	20.6	1.0	1	0.002	0.027	$< 10^{-3}$	0.009
5000	8.4	42.2	1.0	1	0.001	0.013	$< 10^{-3}$	0.006
6000	12.6	71.6	1.0	1	0.001	0.013	$< 10^{-3}$	0.011
7000	16.4	116.2	1.0	1	0.001	0.011	$< 10^{-3}$	0.007
8000	20.2	65.3	1.0	1	0.001	0.010	$< 10^{-3}$	0.003
9000	26.5	238.4	1.0	1	0.001	0.007	$< 10^{-3}$	0.006
10000	32.1	197.2	1.0	3	0.001	0.007	$< 10^{-3}$	0.007
15000	79.5	1858.2	1.2	21	$< 10^{-3}$	0.005	$< 10^{-3}$	0.008
20000	139.0	2957.9	1.5	55	$< 10^{-3}$	0.004	$< 10^{-3}$	0.004
25000	204.5	3981.7	1.8	57	$< 10^{-3}$	0.003	$< 10^{-3}$	0.001
30000	274.8	3491.1	2.1	81	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001

Table 2: Performances of the enumerative algorithm.

The columns NODES refer to nodes of the enumeration scheme, i.e. nodes where the reduction procedure was applied, then either branch or ILP solution occurred (the related instance of (1)–(3) presented no more than $1.4 \cdot 10^7$ nonzeros). Columns $UB - GAP$ and $LB - GAP$ indicate the percentage relative deviations $\frac{UB-OPT}{OPT}$ and $\frac{OPT-LB}{OPT}$ of upper and lower bounds at the root node from the optimal solution value. For $n = 1000, 2000, 3000, 4000$, the CPU times (both average and worst-case) were drastically reduced with respect to those in Table 1. Also, we note that for n up to 9000 jobs, the enumeration scheme only needed to handle the root node.

The proposed algorithm exhibits extremely good performances also when applied to the deadline-free special case $1|\ |\sum w_i U_i$ with the results summarized in Table 3. For this case, the algorithm solves to optimality all instances with up to $50,000$ jobs, strongly outperforming the state of the art algorithms (see [8]).

5 Conclusions

We have proposed for the $1|\bar{d}_i|\sum w_i U_i$ problem an exact procedure that is able to solve to optimality very large size instances by exploiting a compact ILP formulation of the problem. As an outcome of this work, we remark that, also for scheduling problems, whenever structural properties allow to derive "good" LP/ILP formulations, then already commercial solvers standalone can handle reasonably large size instances. Moreover, adding minimal additional procedures that rely on some

	CPU Time (s)		NODES		UB-GAP$_{\%}$		LB-GAP$_{\%}$	
n	avg	max	avg	max	avg	max	avg	max
1000	0.6	4.1	1.0	1	0.003	0.069	0.001	0.037
2000	1.2	4.5	1.0	1	0.001	0.028	$< 10^{-3}$	0.008
3000	2.0	5.8	1.0	1	0.001	0.022	$< 10^{-3}$	0.005
4000	3.3	8.8	1.0	1	0.001	0.022	$< 10^{-3}$	0.005
5000	4.9	11.1	1.0	1	$< 10^{-3}$	0.009	$< 10^{-3}$	0.004
10000	19.3	211.7	1.0	1	$< 10^{-3}$	0.005	$< 10^{-3}$	0.007
15000	37.3	84.4	1.0	1	$< 10^{-3}$	0.003	$< 10^{-3}$	0.001
20000	61.2	233.3	1.0	7	$< 10^{-3}$	0.003	$< 10^{-3}$	0.005
25000	93.9	459.7	1.1	11	$< 10^{-3}$	0.002	$< 10^{-3}$	0.003
30000	128.4	193.8	1.0	1	$< 10^{-3}$	0.001	$< 10^{-3}$	0.001
35000	177.1	1671.6	1.4	59	$< 10^{-3}$	0.001	$< 10^{-3}$	0.002
40000	215.6	966.0	1.2	43	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001
45000	281.1	537.0	1.0	7	$< 10^{-3}$	0.001	$< 10^{-3}$	0.002
50000	315.3	736.0	1.2	17	$< 10^{-3}$	0.002	$< 10^{-3}$	0.001

Table 3: Performances of the enumerative algorithm on instances of the deadline-free $1|\ |\sum w_i U_i$ problem.

problem structure can greatly boost performances. These results emphasize the need of research on novel ILP formulations for other more complex scheduling models.

References

[1] E.L. Lawler (1983), Scheduling a single machine to minimize the number of late jobs Report CSD-83-139, EECS Department, University of California, Berkeley. Available from `http://techreports.lib.berkeley.edu`.

[2] R.M. Karp (1972), Reducibility among Combinatorial Problems, in *Complexity of Computations,*, R.E. Miller and J.W. Thatcher (Eds.), *Plenum Press, New York*, 85 – 103.

[3] J.M. Moore (1968), An n job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Science* **15**, 102 – 109.

[4] J.K. Lenstra and A.H.G. Rinnooy Kan and P. Brucker (1977), Complexity of machine scheduling problems, *Annals of Discrete Mathematics* **1**, 343 – 362.

[5] C.N. Potts and L.M. Van Wassenhove (1988), Algorithms for scheduling a single machine to minimize the weighted number of late jobs, *Management Science* **34**, 7, 843 – 858.

[6] A.M.A. Hariri and C.N. Potts (1994), Single machine scheduling with deadlines to minimize the weighted number of tardy jobs *Management Science* **40**, 12, 1712 – 1719.

[7] Ph. Baptiste, C. Le Pape and L. Péridy (1998), Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraint *LNCS (Proc. of CP)* **1520**, 87 – 101.

[8] R. M'Hallah and R.L. Bulfin (2003), Minimizing the weighted number of tardy jobs on a single machine, *European Journal of Operational Research* **145**, 1, 45 – 56.

[9] S. Dauzère-Pérès and M. Sevaux (2003), Using Lagrangean Relaxation to Minimize the Weighted Number of Late Jobs on a Single Machine, *Naval Research Logistics* **50**, 273 – 288.

[10] S. Dauzère-Pérès and M. Sevaux (2004), An exact method to minimize the number of tardy jobs in single machine scheduling, *Journal of Scheduling* **7**, 405 – 420.

[11] R. M'Hallah and R.L. Bulfin (2007), Minimizing the weighted number of tardy jobs on a single machine with release dates, *European Journal of Operational Research* **176**, 727 – 744.

[12] J.T. Linderoth and M.W.P. Savelsbergh (1999), A computational study of search strategies for mixed integer programming, *INFORMS Journal on Computing* **11**, 2, 173 – 187.

[13] V. T'kindt and F. Della Croce and J.-L. Bouquard (2007), J.-L. Enumeration of Pareto optima for a flowshop scheduling problem with two criteria, *INFORMS Journal on Computing* **19**, 1, 64 – 72.

Order Acceptance and Scheduling Decisions in Make-to-Order Systems

Zehra Bilgintürk, Ceyda Oğuz, Sibel Salman
Koç University, Rumelifeneri Yolu, 34450, Sarıyer, Istanbul, Turkey, {zbilginturk, coguz, ssalman}@ku.edu.tr

In this paper, we examine simultaneous order acceptance and scheduling decisions in a make-to-order system. We model the manufacturing environment as a single machine environment, with a set of orders coming from customers. In this pool of orders, of which we know the release dates, due dates, processing times, deadlines, sequence dependent setup times and revenues, manufacturer has to decide on the subset of orders to select, considering the limited production capacity in order to maximize the profit. The tardiness of orders is penalized and revenue gained from an order decreases with tardiness, until the deadline, after which no revenue can be obtained. The problem generalizes some well-known scheduling problems with the objective of minimizing total tardiness and has the sequence dependent setup times as a further complicating property, as well as the order acceptance decisions. As a result, the problem is NP-hard. In this study, first, we give an MILP model for the problem, and solve it with a commercial solver. Next, we propose a Simulated Annealing algorithm to solve the large-size problems. We compare the performance of the algorithm with respect to the optimal solution, when the problem can be solved optimally. We use upper bounds generated by LP relaxation for comparison when optimal objective values are not available.

Keywords: Production Scheduling, Meta-heuristic Search.

1 Introduction

Make-to-order firms, such as a custom-made furniture shop or a boat builder, manufacture customized products, in which the production is initiated by a customer order and, typically, no work-in-process inventory is carried for this kind of products. The manufacturer will gain a revenue if a particular order is accepted and manufactured by using the production resources. Since the main production resource, i.e., the capacity is limited, manufacturing one order may delay another one. If there are no deadlines, then the manufacturer will just incur some penalty for the orders which are delayed. If there are deadlines for the orders, such delays may lead to rejecting customer orders. In a competitive make-to-order environment, a manufacturer should use the capacity efficiently, satisfy the customers' expectations at a high level and gain the maximum revenue from the incoming orders. Therefore, the manufacturer should find a balance between the revenue that can be gained from the accepted orders and the cost incurred by manufacturing these orders. So the question is which orders to accept to maximize the profit considering the limited production capacity of the firm. Accepting an order whose deadline cannot be met causes a loss in the reputation of the firm as well as in the revenue. To avoid such circumstances, the order acceptance decisions should be made very carefully. When high utilization levels are present, firms accept the orders that will bring high revenues and have comparably less production capacity requirements only. If an order will be tardy, that is, if its completion time will exceed its due date, the manufacturer has to sacrifice from some of its revenue, if this order is accepted.

In this study we consider the problem from the manufacturer's point of view and try to maximize the profit gained from the accepted orders. The profit is defined as the total revenues minus the total weighted tardiness. The manufacturing environment consists of a single machine with a

limited capacity and a set of independent orders O ($|O| = n$) at the beginning of the planning period. For each order $i \in O$, we know the data including the release date r_i after which the order is available, the processing time w_i, the due date d_i up to which order i can be produced without incurring a penalty, the deadline $\bar{d}_i$ after which order i cannot be produced ($\bar{d}_i \geq d_i$), the sequence dependent setup time s_{ij} which is required when order i is scheduled before order j, the revenue e_i to be gained if the order is accepted. The revenue e_i may also reflect the level of priority or the importance of order i. The decision to be made in this environment includes determining the set of accepted orders and the corresponding optimal schedule for this set of orders to minimize the weighted tardiness. The tardiness of order i is defined as $T_i = \max\{0, C_i - d_i\}$ where C_i is the completion time of order i.

As our aim is to maximize the profit gained from the accepted orders, which is defined as the total revenues minus the total weighted tardiness, our objective function is then defined by $\max \sum_{i \in A} P_i = \sum_{i \in A} (e_i - T_i \times \alpha_i)$ where A is the set of accepted orders, α_i is the weight for the tardiness of order i and P_i is the profit gained from the accepted order i. The profit P_i reflects the fact that if an order i is tardy, the customer will receive a discount. However we note that the early delivery is not penalized as the manufacturer is assumed to have unlimited capacity for finished product storage. In this setting, the manufacturer has the right to reject an order that will not be profitable. We will denote this problem as OASP (Order Acceptance and Scheduling Problem) throughout the paper.

OASP is a common real-life problem faced at the production companies as well as the service companies. One such example to manufacturing companies facing this problem is a packaging firm that has varying sequence dependent setup times for each order. Consulting firms is an example of service companies working on a make-to-order basis.

The relevant research to OASP can be categorized under two subtitles: 1) studies concerning the total tardiness problem, since OASP is a generalization of the single machine total tardiness problem, and 2) studies concerning the simultaneous order acceptance and scheduling problem.

The total tardiness problems have been studied since 1960s. Emmons[1] is one of the first researchers working on the total tardiness problems. Du and Leung [2] proved that minimizing total tardiness in one machine is NP-hard in the ordinary sense. The addition of sequence dependent setup times increases the veracity of the problem, but it increases the computational complexity as well. Hence, the total tardiness problem with sequence dependent setup times (TTSDS) is also NP-hard. Comprehensive literature reviews on setup considerations in scheduling problems are presented by Allahverdi et al. [3] and Yang et al. [4].

Application of metaheuristics as well as branch-and-bound algorithm to the TTSDS is common. Rubin and Ragatz [5] applied a genetic search to the problem. The success of simulated annealing technique in combinatorial optimization problems motivated researchers to apply this method to TTSDS problem. Tan and Narasimhan [6] used a Simulated Annealing approach to TTSDS problem. The algorithm proposed by the authors is not suitable for OASP since OASP contains additional complicating issues such as the order acceptance, the release dates, and the deadlines.

A recent study on order acceptance and scheduling problem is presented by Slotnick and Morton [7]. OASP is a generalization of this problem as it takes the sequence dependent setup times into account. Also OASP problem fills a gap in the scheduling literature since make-to-order environments with the sequence dependent setup times and strict deadlines are not covered.

In the remaining parts of the paper, we will first present an MILP model for OASP. As our computational experiments showed that it is not possible to solve this model with a branch-and-bound algorithm if $n > 10$, we then propose a Simulated Annealing algorithm to solve the problem with large-size instances. We conclude the paper with the results of our computational experiments

in which the performance of the Simulated Annealing algorithm is analyzed.

2 Mathematical Model

We formulate the order acceptance and sequencing problem as a mixed integer problem below. In this model, we define dummy orders, order 0 and order $n+1$, where definitely, order 0 is assigned to the first position, and order $n+1$ is assigned to the last position, whatever the sequence of orders in between is. There are two sets of binary variables in the model. The first one, y_{ij}, determines the sequence of orders, and the second one, I_i, determines which orders are accepted. Hence, $y_{ij} = 1$ if order i precedes order j, and $y_{ij} = 0$ otherwise; and $I_i = 1$ if order i is selected, and $I_i = 0$ otherwise. The objective function of the model is to maximize the total profit gained from accepted orders.

Constraint sets (1) and (2) states that, if an order is accepted, this order precedes only one job and, is succeeded by only one job. If it is not accepted, it does not take place in the sequence. These constraints also prohibit the preemption of the jobs. Constraint set (3) implies that, if order j is preceded by order i, then the completion time of order j should be larger than the completion time of order i plus the sequence dependent setup time between orders i and j, plus the processing time of order j. If order i does not precede order j in the sequence, $C_j \geq 0$ should be the only limit, and this constraint ensures this outcome. Constraint set (4) states that, if order j is accepted, and is preceded by order i in the schedule, then the completion time of order j should be greater than the release date of that order plus the sequence dependent setup time between orders i and j, plus the processing time of order j. In case where order i does not precede order j, the completion time of order j has looser bounds which are given by the release time of order j and its deadline. In case where order j is not accepted, it will not be processed so, $C_j = 0$. These constraints enable us to calculate the correct completion times of the orders. Constraint sets (5) and (6) are very general constraints and they put bounds on C_i and T_i. Set (7) calculates the maximum tardiness for an accepted order. Sets (8) and (10) ensure nonnegativity of P_i and T_i, implied by the definitions of these decision variables. Constraint set (9) calculates the profit manufacturer can gain, when order i is accepted and incurs tardiness of T_i. Here, the profit from an accepted order decreases as this order becomes tardy and becomes 0 at its deadline. α_i coefficient is chosen accordingly to satisfy this assumption. For one unit time of tardiness, revenue decreases by α_i units. α_i coefficient is calculated by the formula $e_i/(\bar{d}_i - d_i)$ for each order i. Sets (11) and (12) include some necessary initializations for dummy nodes 0 and $n+1$. Constraint set (13) defines the decision variables of the model as binary variables.

When we solve this MILP by CPLEX solver, it is observed that the optimal solution can be obtained only up to $n = 10$ orders, which is not surprising. Hence we resort to metaheuristic algorithms as they were successfully applied to TTSDS problem. In this study we chose Simulated Annealing and analyzed its performance.

MILP:

$$\max \sum_{i=1}^{n} P_i$$

$$s.t \quad \sum_{j=1,j\neq i}^{n+1} y_{ij} = I_i, \qquad \forall i = 0,...,n \tag{1}$$

$$\sum_{j=0,j\neq i}^{n} y_{ji} = I_i, \qquad \forall i = 1,...,n+1 \tag{2}$$

$$C_i + (s_{ij} + w_j) \times y_{ij} + \bar{d}_i \times (y_{ij} - 1) \leq C_j, \qquad \forall i = 0,...,n, \forall j = 1,...,n+1, i \neq j \tag{3}$$

$$r_j \times I_j + (s_{ij} + w_j) \times y_{ij} \leq C_j, \qquad \forall i = 0,...,n, \forall j = 1,...,n+1, i \neq j \tag{4}$$

$$T_i \geq C_i - d_i, \qquad \forall i = 0,...,n+1 \tag{5}$$

$$C_i \leq \bar{d}_i \times I_i, \qquad \forall i = 0,...,n+1 \tag{6}$$

$$T_i \leq (\bar{d}_i - d_i) \times I_i, \qquad \forall i = 0,...,n+1 \tag{7}$$

$$T_i \geq 0, \qquad \forall i = 0,...,n+1 \tag{8}$$

$$P_i \leq e_i \times I_i - T_i \times \alpha_i, \qquad \forall i = 1,...,n \tag{9}$$

$$P_i \geq 0, \qquad \forall i = 1,...,n \tag{10}$$

$$C_0 = 0,\ C_{n+1} = \max_{i=1,...,n} \{\bar{d}_i\}, \qquad \forall i = 1...n \tag{11}$$

$$I_0 = 1,\ I_{n+1} = 1 \tag{12}$$

$$I_i \in \{0,1\}, y_{ij} \in \{0,1\}, \qquad \forall i = 1,...,n \tag{13}$$

3 Simulated Annealing Algorithm

In this section we describe the Simulated Annealing algorithm for solving the OASP. Simulated Annealing is a heuristic method that uses thermodynamic concepts. Simulated Annealing has a very important aspect in its nature: it is good at diversifying and less likely to get stuck in the local optimum than other metaheuristics. Simulated Annealing accepts a new solution even it is worse than before, with a probability given by $\exp(-\delta f/T)$, where δf is the increase in the objective function value and T is the current temperature. Simulated Annealing technique was successfully applied to combinatorial optimization problems in the literature as in [6]. Our algorithm is described below.

1. Read the input data.
2. Set the control parameters:
   (a) Initial temperature (T_{max})
   (b) Temperature decay rate, θ
   (c) Set $T_{current} = T_{max}$
3. Construct the initial solution (not necessarily feasible).
   (a) Sequence the jobs in descending order of $slacktime_j = \bar{d}_j - r_j - (minsetup_j + w_j)$;
   (b) Calculate the completion time C_j, the tardiness T_j and the revenue gained, P_j.

(c) Count the number of orders violating their deadlines d_j.

4. While number of violating orders > 0 do the following:

   (a) Perform the following loop ITER times:

      i. Generate two different random integers between 1 and n. (n is the number of available orders)

      ii. Exchange the orders having these indices.

      iii. Calculate C_j's, T_j's and P_j's and the number of violating orders in this new sequence.

         A. if (Total revenue $>$ best revenue), accept the new sequence.

         B. if (Total revenue $\leq$ best revenue),

            - Calculate the probability of accepting, p=exp(-(-Total revenue+best revenue)/$T_{current}$).
            - Select uniformly distributed random number m, from the interval (0,1).
              - if $m < p$, accept the sequence.
              - if $m \geq p$ reject the new sequence, and continue with the former best sequence.

         C. Return to Step (i).

   (b) Set $T_{current} = \theta \times T_{current}$

   (c) Calculate the ratio $ratio_j = e_j/(w_j + minsetup_j)$ for violating orders.

   (d) Reject the order having the smallest ratio.

   (e) Return to Step (4)with new sequence.

5. If number of violating orders=0, i.e., a feasible solution is obtained, perform the following for ITER1 times:

   (a) Generate two different random integers between 1 and n. (n is the number of available orders)

   (b) Exchange the orders having these indices.

   (c) Calculate C_j's, T_j's and P_j's and the number of violating orders in this new sequence.

      - Calculate the probability of accepting, p=exp(-(-Total revenue+best revenue)/$T_{current}$).
      - Select uniformly distributed random number m, from the interval (0,1).
        - if $m < p$, accept the sequence.
        - if $m \geq p$ reject the new sequence, and continue with the former best sequence.

4 Computational Results

In order to analyze the performance of the Simulated Annealing algorithm (SA), we performed a computational experiment. In this section we first describe the data generation together with the experiments and then present the results.

In this study, release dates r_i, processing times w_i, sequence dependent setup times s_{ij}, and revenues e_i are generated from a uniform distribution from the interval of [0,$w_T \times \tau$], [1,20], [1,10], and [1,20], respectively. The due dates are generated so that for each order i, due date d_i is in the interval of $(w_T + \max_j s_{ij} + r_i) \times [1 - \tau - R/2, 1 - \tau + R/2]$ and integer, where τ, R and w_T represent the tardiness factor, the due date range and the total processing time of all orders respectively. The deadlines are generated from the formula $\bar{d_i}$=$d_i + R \times w_i$. The weight α_i is calculated by the

formula α_i=$e_i/(\bar{d}_i - d_i)$. In this setting, revenue gained from an order i becomes 0 at its deadline $\bar{d}_i$.

The first part of the computational study compares the performance of SA with respect to the optimal objective values. The MILP model was coded in ILOG and solved by CPLEX solver. Time limit was 5000 seconds for the CPLEX runs. The SA was coded in JAVA. All of the runs were taken on a computer with a Pentium 4 processor, 2.4 GHz speed, and 2 GB of RAM. We present the maximum, minimum, and average % deviation of the SA objective function value from the optimal objective function value and the corresponding CPU times in Table 1. For each parameter combination, we solved 5 instances of OASP, generated randomly as explained above, and reported the results. From these results, it is seen that the hardest problem instance for CPLEX is generated when $\tau = 0.1$ and $R = 0.1, 0.3$. For the problem sizes above 10, it is impossible to find the optimal solution within the time limit of 5000 seconds in this setting. The results of the experiments confirm that, as τ increases, the problem gets less time consuming for the optimal method, but more time consuming for the SA. In addition, when we keep τ constant, and increase R, we see that the problem becomes easier for the optimal method. We found that the SA objective function values are competitive with the optimal objective function values for $n = 10$ which can be observed from the average percentage deviations displayed in Table 1. SA algorithm is clearly a fast and a good quality metaheuristic for $n = 10$.

Table 1: SA and optimal solution comparison for the OASP with problem sizes of 10.

			% Deviation			CPU Time (SA)			CPU Time (CPLEX)		
n	τ	R	Max	Min	Avg	Max	Min	Average	Max	Min	Average
10	0.1	0.1	6%	0%	2%	35.62	30.98	32.36	> 5000.00	> 5000.00	> 5000.00
		0.3	10%	2%	6%	42.06	32.41	34.46	> 5000.00	> 5000.00	> 5000.00
		0.5	0%	0%	0%	51.08	22.94	29.85	> 5000.00	190.43	2199.06
		0.7	0%	0%	0%	23.84	22.98	23.45	2101.82	7.82	426.38
		0.9	0%	0%	0%	39.39	23.16	26.78	3545.00	2.00	1314.93
	0.3	0.1	11%	0%	4%	136.67	47.98	104.94	> 5000.00	1430.73	3569.27
		0.3	12%	3%	6%	199.60	73.57	118.18	> 5000.00	2390.90	3546.89
		0.5	4%	0%	2%	130.41	77.13	110.41	3111.22	878.61	2027.17
		0.7	8%	0%	2%	83.93	23.35	45.15	> 5000.00	58.29	2067.75
		0.9	1%	0%	0%	38.89	23.17	29.71	2419.26	35.67	932.12

As it is not possible to obtain the optimal solution for OASP when $n > 10$, we compared the performance of the SA with respect to an upper bound in the second part of the computational study. An easy to compute upper bound is found by the LP relaxation of MILP. For each parameter combination of the problem, we solved 10 instances generated randomly as described above and reported the results. We present the maximum, minimum, and average % deviation of the SA objective function value from the objective function value of the LP relaxation and the corresponding CPU times in Table 2. We carried out the experiments for problem sizes of 10, 15, 20, 25, and 50. LP relaxation is observed to be relatively tight when tardiness factor τ is equal to 0.1 and 0.3 for n=10 (average SA-LP gaps are around 1% and 9%, respectively). LP relaxation solution times are not included in the table since the problem is solved within maximum of 3–4 seconds when integrality restrictions are discarded.

We can make the following observations from Table 2:

- The CPU times of SA are reasonable even for large-size problem instances.
- The average % deviation is below 25% in all cases.

- The observation we made for small-size problem instances regarding the difficult problem cases is not clear as in Table 1 because when we analyze the CPU times of SA, we cannot see much difference depending on different cases. However, we observe that the average % deviation of the difficult cases for small-size problems is large in Table 1 as well. When we examine the results in Table 2, we see that the differences among the average % deviations for different cases are not so large. Hence, we may conclude that for large-size problems, the difficulty of the problem remains the same for different R values.
- We observe that as τ increases, the problem gets more difficult for SA.
- The large % deviations can be attributed to the difficulty of the problem, which results in loose upper bounds from LP relaxations.

Overall we can say that SA is an acceptable solution method for OASP even for large-size problems as its average % deviation from the LP relaxation is below 25%.

5 Summary and Conclusions

In this study, we have studied a problem of simultaneous order acceptance and scheduling decisions in a make-to-order system and have provided preliminary results obtained from the computational experiments. First, we developed an MILP model for the problem. We obtained the optimal solutions for small-size problems (i.e, n is up to 10) and observed that when the problem size is greater than 10, the problem becomes intractable in a reasonable time limit. For this reason, we then proposed a Simulated Annealing algorithm for the problem. The results of the computational experiments showed that the Simulated Annealing algorithm finds good solutions compared to upper bounds which were found by LP relaxation in reasonable time limits for large-size problems.

As the next step of our study, we will continue the computational analysis and investigate the difficulty of the problem for other values of τ.

References

[1] H. Emmons (1969), One machine sequencing to minimize the certain functions of job tardiness, *Operations Research* **17**, 701 – 715.

[2] J. Du and J. Leung (1990), Minimizing total tardiness on one machine is NP-Hard, *Mathematics of Operations Research* **15**, 483 – 495.

[3] A. Allahverdi, J.N.D. Gupta and T. Aldowaisan (1999), A review of scheduling research involving setup considerations, *Omega* **27**, 219 – 239.

[4] W. Yang and C. Liao (1999), Survey of scheduling involving setup times, *International Journal of Systems Science* **30**, 143 – 155.

[5] P.A. Rubin , G.L. Ragatz (1995), Scheduling in a sequence dependent setup environment with genetic search, *Computers and Operations Research* **22**, 85 – 99.

[6] K.C. Tan and R. Narasimhan (1997), Minimizing tardiness on a single processor with sequence dependent setup times: A simulated annealing approach, *Omega* **25**, 619 – 634.

[7] S.A. Slotnick and T.E. Morton (2007), Order acceptance with weighted tardiness, *Computers and Operations Research* **forthcoming**.

Table 2: SA and LP relaxation comparison for the problem with different sizes.

			% Deviation			CPU Time (SA)		
n	τ	R	Max	Min	Avg	Max	Min	Average
10	0.1	0.1	7%	1%	3%	79.84	30.98	43.77
		0.3	10%	0%	3%	42.86	32.41	42.11
		0.5	3%	0%	0%	51.08	22.94	35.77
		0.7	0%	0%	0%	41.78	22.98	33.66
		0.9	5%	0%	1%	41.69	23.16	34.20
	0.3	0.1	25%	2%	15%	136.67	47.98	97.44
		0.3	24%	8%	14%	199.60	63.69	98.81
		0.5	17%	1%	7%	130.41	60.91	92.65
		0.7	13%	0%	6%	83.93	23.35	50.33
		0.9	16%	0%	3%	61.61	23.17	37.95
15	0.1	0.1	15%	3%	8%	109.93	83.62	90.56
		0.3	16%	1%	5%	87.48	63.53	75.61
		0.5	6%	0%	3%	82.59	45.71	63.00
		0.7	3%	0%	1%	87.06	45.48	55.81
		0.9	10%	0%	2%	106.48	45.78	60.96
	0.3	0.1	21%	10%	16%	154.93	117.27	140.36
		0.3	23%	4%	14%	151.39	105.89	126.02
		0.5	17%	6%	11%	126.48	90.42	108.72
		0.7	19%	3%	10%	127.95	85.46	109.74
		0.9	16%	2%	9%	145.36	66.11	106.40
20	0.1	0.1	11%	3%	7%	140.92	114.92	126.51
		0.3	7%	1%	4%	119.35	93.17	103.63
		0.5	4%	0%	2%	118.21	56.85	85.63
		0.7	5%	0%	2%	113.89	51.82	76.85
		0.9	10%	0%	3%	116.39	49.00	73.43
	0.3	0.1	26%	10%	18%	209.56	149.82	190.16
		0.3	26%	11%	16%	204.43	154.21	181.14
		0.5	19%	6%	14%	230.00	144.76	167.63
		0.7	18%	6%	14%	203.40	139.40	160.72
		0.9	29%	6%	13%	182.89	136.95	154.27
25	0.1	0.1	15%	5%	8%	197.95	143.79	179.01
		0.3	9%	3%	6%	172.50	124.53	146.98
		0.5	9%	2%	5%	106.87	139.86	122.92
		0.7	7%	0%	4%	54.77	125.03	105.93
		0.9	8%	0%	3%	133.92	76.72	99.20
	0.3	0.1	23%	11%	19%	288.22	243.44	260.58
		0.3	21%	13%	17%	288.42	211.38	246.88
		0.5	23%	8%	15%	265.49	197.74	231.86
		0.7	31%	9%	16%	263.27	196.89	229.27
		0.9	28%	7%	14%	289.56	199.17	228.99
50	0.1	0.1	12%	7%	10%	555.13	413.41	499.36
		0.3	10%	5%	8%	527.08	383.69	455.88
		0.5	13%	2%	9%	492.39	337.47	433.54
		0.7	14%	6%	8%	516.08	345.20	430.44
		0.9	12%	4%	10%	591.00	326.92	434.34
	0.3	0.1	28%	19%	25%	854.69	669.08	771.91
		0.3	28%	16%	21%	907.72	683.11	747.74
		0.5	25%	17%	22%	794.33	690.00	757.54
		0.7	29%	19%	23%	847.30	709.92	785.50
		0.9	33%	17%	25%	824.16	642.45	759.00

Scheduling with Special Case of Multipurpose Machines

Mourad Boudhar, Hamza Tchikou
Faculty of Mathematics, University USTHB, BP 32 Bab-Ezzouar, El-Alia 16111, Algiers, Algeria,
{mboudhar, tch_hamza}@yahoo.fr

We consider the problem of minimizing the makespan on m special multipurpose machines (called ordered parallel machines) $M_1, \ldots, M_m$ in which the execution of each job J_i $(1 \leq i \leq n)$ requires a time p_i and a machine among a subset $M_{h_i}, \ldots, M_m$ of machines. We prove the NP-hardness of the general problem and present some polynomial subproblems. Heuristics with an exact algorithm of branch and bound type are also presented with numerical experimentations.

Keywords: Complexity of Scheduling Problems, Heuristic Search, Machine Scheduling.

1 Introduction

We consider the problem of scheduling a set of jobs by allowing ordered machines (special case of multipurpose machines) to the jobs and fixing their starting times. The set of machines is denoted by $M = \{M_1, M_2, \ldots, M_m\}$, where m represents the number of machines and the set of jobs is denoted by $J = \{J_1, J_2, \ldots, J_n\}$, where n represents the number of jobs. The execution of each job J_i $(1 \leq i \leq n)$ requires a processing time p_i and a machine among a subset of machines $\{M_{h_i}, \ldots, M_m\}$ $(h_i \in \{1, \ldots, m\})$. The objective is to minimize the length of the scheduling (makespan) denoted by $C_{max} = \max_{1 \leq i \leq N} \{C_i\}$ where C_i is the completion time of the job J_i.

As applications of the multipurpose machines cover a large class of industrial problems, our case considers one particular type of these problems which is the execution of a job requiring a machine among a subset of machines, such as if the job J_i $(1 \leq i \leq n)$ is candidate to be processed on the machine M_k it is also candidate to be processed on all the subset $\{M_k, M_{k+1}, \ldots, M_m\}$. Example: Consider m machines $M_1, M_2, \ldots, M_m$ of guillotine type, where a machine M_j can cut only sheets having a given maximum length (called a threshold). Let us suppose that the machines are arranged in non decreasing order of their thresholds, therefore if a sheet can be cut on the third machine it can be also processed on the fourth, the fifth, ..., the m-th machine.

The type of machines considered here is a subclass of the multipurpose machines. In a model of scheduling with multipurpose machines, denoted in the literature by " MPM ", there exists a subset of machines $\mu_i \subseteq \{M_1, \ldots, M_m\}$ associated to the job J_i such as this one is processed on one of the machines of the subset μ_i. If the multipurpose machines are parallel identical (resp. uniform), the corresponding situation is denoted by $PMPM$ (resp. $QMPM$). One denotes P (resp. Q) the special case of $PMPM$ (resp. $QMPM$) where all the subsets μ_i contain all the m machines.

This article is organized as follows. In section 2 we give an analysis of the complexity of the problem. The section 3 gives some polynomial subproblems. An exact method is given in section 4 and some heuristics are given in section 5. Numerical experimentations are realized in section 6. A conclusion ends this paper.

2 Formulation and complexity

We start by proposing a mathematical formulation of the problem in the form of a linear program in binary variables. With this intention, we define the following variables and mathematical constraints: Let x_{ij} be the variables which indicate if the job J_i is processed on the machine M_j, therefore

$$x_{ij} = \begin{cases} 1 & if \quad J_i \text{ is processed on } M_j \\ 0 & if \quad no \end{cases}$$

for $i = 1, \ldots, n$ and $j = h_i, \ldots, m$

And let y be the variable which indicates the completion time of the set of the jobs ($y = C_{max}$).

The mathematical model is written as follows:

$$\min\ y$$

$$\text{s.t.} \begin{cases} \sum\limits_{i:h_i \leq j} x_{ij} p_i \leq y & for \quad j = 1, \ldots, m \\ \sum\limits_{h_i \leq j \leq m} x_{ij} = 1 & for \quad i = 1, \ldots, n \\ x_{ij} \in \{0, 1\} & for \quad i = 1, \ldots, n \ ; \ j = h_i, \ldots, m \\ y \geq 0 \end{cases}$$

Let us denote by $MPMO2$ (parallel multi-purpose machines ordered), the problem of two multi-purpose parallel machines, such that if a job J_i ($i = 1, \ldots, n$) can be processed on a machine M_1, it can also be processed on the machine M_2 ; the inverse is not true. The problem $MPMO2//C_{max}$ is NP-hard because the problem $P2//C_{max}$, which is a particular case ($h_i = 1$ for all the jobs) is NP-hard. We will now show that the problem remains NP-hard even if the number of jobs to be processed on the second machine is constant and known in advance and all these jobs have an identical processing time.

Theorem 1 *The problem $MPMO2//C_{max}$ is NP-hard even if the number of jobs processed on the second machine is fixed, and all these jobs have an identical processing time.*

Remark 1 *As the problem $MPMO2//C_{max}$ (with only two machines) is NP-hard, then the problem $MPMO//C_{max}$ in its general form is also NP-hard.*

3 Some polynomial problems

In this section we will give some polynomial subproblems and some algorithms of resolution in polynomial time.

Problem $PMPMO/p_i = 1/C_{max}$

For this problem the fact that the processing time of the jobs is equal to 1, allows us to propose a polynomial time algorithm to solve it.

Algorithm A1 ;
begin $y := 0$; $nb := 0$;
    for $j := m$ down 1
    do - $X_j := \{J_i \in J / h_i = j\}$;
        - if $X_j \neq \emptyset$
            then - $J := J \setminus X_j$; $nb := nb + |X_j|$; $y := max\{y, \lceil nb/(m - j + 1) \rceil\}$;
                - schedule the jobs of X_j on the machines $M_j, \ldots, M_m$ with a finish time equal to y ;
            endif
    enddo ;

$C_{max} := y$

end.

At each iteration, X_j contains the jobs where $h_i = j$; nb is the number of scheduled jobs and y is the completion time of processing of the scheduled jobs.

Example 1 *Let us process* 9 *jobs* $J_1, \ldots, J_9$ *with processing time equal to* 1 *on three machines* M_1, M_2 *and* M_3 *with* $h_1 = h_2 = h_3 = h_4 = h_5 = 1$, $h_6 = h_7 = h_8 = 2$ *and* $h_9 = 3$ *in minimal time.*

The execution of this algorithm gives:
First iteration: $j = 3$; $X_3 = \{J_9\}$; $nb = 1$; $y = max\{0, \lceil 1/1 \rceil\} = 1$
Second iteration: $j = 2$; $X_2 = \{J_6, J_7, J_8\}$; $nb = 4$; $y = max\{1, \lceil 4/2 \rceil\} = 2$
Third iteration: $j = 1$; $X_1 = \{J_1, J_2, J_3, J_4, J_5\}$; $nb = 9$; $y = max\{2, \lceil 9/3 \rceil\} = 3$
This gives $C_{max} = 3$. An optimal schedule is given on figure 1.

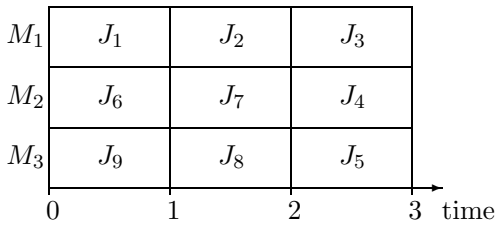


Figure 1: Solution of example 1

Theorem 2 *The algorithm A1 solves the problem* $MPMO/p_i = 1/C_{max}$ *and in* $O(nlogn)$.

Scheduling on uniform machines $QMPMO/p_i = 1/C_{max}$

In this case, each machine M_j $(1 \leq j \leq m)$ has a speed S_j $(S_j \geq 1)$. The processing time of a job is, therefore, a standard processing time, i.e. that of a machine having a speed of processing equal to 1.

For this problem an approach of resolution can be the test of admissibility via the approach flow in a network. We build the network with two sets in the following way:

- The first set is that of the jobs of J.
- The second set is that of the machines of M.
- Each job J_i of J is connected to the source by an arc of capacity equal to 1.
- Each machine M_j of M is connected to the sink by an arc of capacity equal to b_j
- If J_i can be processed on machine M_j then J_i is connected to M_j by an arc of capacity equal to 1.

It is seen that to find a maximum flow equal to n on the network with $b_j = n$ for $(1 \leq j \leq m)$, corresponds to the construction of a realizable scheduling where all the jobs are assigned to the various machines.

Algorithm A2 ;
begin $x := 0$; $y := n \times \max\limits_{1 \leq j \leq m} \{S_j\}$;
while $x < y - 1$
do - $z := \lceil (x + y)/2 \rceil$;
- Apply the flow max. algorithm with $b_j := \lfloor z/S_j \rfloor$ for $j := 1, \ldots, m$;
- If the maximum flow is equal to n then $y := z$ else $x := z$ endif

enddo ;
$C_{max} := \lceil (x+y)/2 \rceil$
end.

$b_j := \lfloor z/S_j \rfloor$ corresponds to the maximum number of jobs which one can process on the machine M_j with makespan $\leq z$.

This algorithm is a dichotomic procedure of search on the interval $[0, n \times \max_{1 \leq j \leq m} \{S_j\}]$, it determines the smallest completion time C_{max} on this interval because $0 < C_{max} \leq n \times \max_{1 \leq j \leq m} \{S_j\}$.

Theorem 3 *The algorithm A2 determines an optimal solution, in* $O(n^3 log n \max_{1 \leq j \leq m} \{S_j\})$, *for problem* $QMPMO/p_i = 1/C_{max}$.

Example 2 *Consider six jobs* $J_1, \ldots, J_6$ *to be processed on three machines* M_1, M_2 *and* M_3 *where the processed speeds of the machines are respectively* 1, 2 *et* 3, *under the constraints* $h_1 = h_2 = h_3 = 1$, $h_4 = h_5 = 2$ *and* $h_6 = 3$.

The algorithm starts with the Construction of the network as figure 2 shows it below.

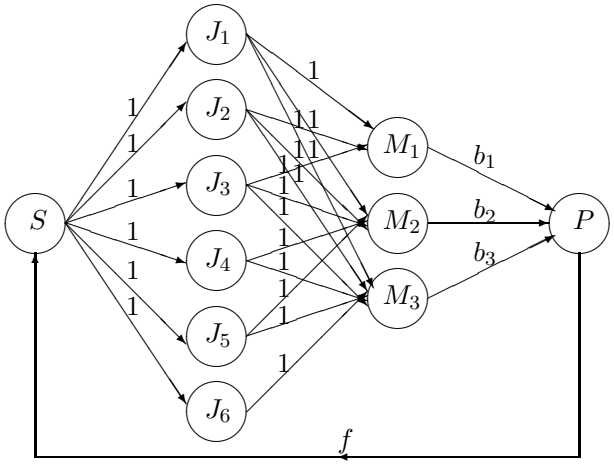


Figure 2: Network of example 2

Initialization : $x := 0$ et $y := n \times \max_{1 \leq j \leq 3} \{S_j\} = 18$
First iteration: $z := 9$; application of the flow max. algorithm with $b_1 = 9$, $b_2 = 4$ and $b_3 = 3$; $f = 6 \Longrightarrow y = 9$
Second iteration: $z := 5$; application of the flow max. algorithm with $b_1 = 5$, $b_2 = 2$ and $b_3 = 1$; $f = 6 \Longrightarrow y = 5$
Third iteration: $z := 3$; application of the flow max. algorithm with $b_1 = 3$, $b_2 = 1$ and $b_3 = 1$; $f = 5 \Longrightarrow x = 3$
Fourth iteration: $z := 4$; application of the flow max. algorithm with $b_1 = 4$, $b_2 = 2$ and $b_3 = 1$; $f = 6 \Longrightarrow y = 4$
Fifth iteration: $y = x + 1$; The algorithm stops .

we stop the loop while with $C_{max} = \lceil (x+y)/2 \rceil = 4$. The optimal solution is given on figure 3.

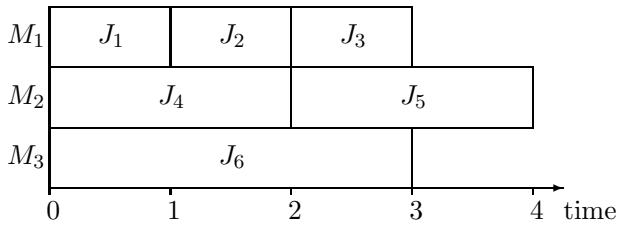


Figure 3: Solution of example 2

Problem $PMPMO/pmtn/C_{max}$

In this part one illustrates a polynomial algorithm for the resolution of the problem $PMPMO/pmtn/C_{max}$. Let us note that the algorithm of Mc Naugthon gives an optimal solution of the problem $P/pmtn/C_{max}$ in $O(n)$.

Now we will propose an algorithm of resolution of the problem $PMPMO/pmtn/\ C_{max}$, by using the Mc Naugthon algorithm as a procedure inserted in the algorithm, and as this procedure is of complexity $O(n^2)$, then its introduction into our algorithm will not negatively influence its complexity.

Let $E = \{z/\exists J_i \in J \ with \ h_i = z\}$.

Algorithm A3 ;
begin - $y := 0$;
- while $E \neq \emptyset$
do - Choose the greatest index of E (let j be this index) ;
- $E := E \setminus \{j\}$; $F_j := \{J_i \in J / h_i = j\}$; $J := J \setminus F_j$;
- $y := \max\{y, \max\limits_{i:h_i \geq j}\{p_i\}, (\sum\limits_{i:h_i \geq j} p_i)/(m-j+1)\}$;
- Apply the Mc Naugthon algorithm to the jobs of F_j and the already scheduled jobs and the machines $M_j, \ldots, M_m$ with $C_{max} = y$;
enddo ;
end.

Theorem 4 *The algorithm A3 gives an optimal solution, in* $O(n^2)$*, for the problem* $PMPMO/pmtn/C_{max}$.

4 Exact Method

In this section we propose an exact algorithm of resolution for the problem of scheduling $PMPMO//C_{max}$ which is certainly not polynomial, under the terms of the complexity of the problem.

Lemma 5 *The value* $P_{max} = \max\limits_{1 \leq i \leq n}\{p_i\}$ *represents a lower bound for the optimal solution of the problem* $PMPMO//C_{max}$.

Lemma 6 *The value* $\max\limits_{j \in E}\{\left\lceil (\sum\limits_{i:h_i \geq j} p_i)/(m-j+1) \right\rceil\}$ *represents a lower bound for the optimal solution of the problem* $PMPMO//C_{max}$.

Therefore according to these two preceding lemmas, one can state the following theorem:

Theorem 7 *For the problem $PMPMO//C_{max}$ we have:*

$$C^*_{max} \geq \max\{\max_{1\leq i\leq n}\{p_i\}, \max_{j\in E}\{\left\lceil (\sum_{i:h_i\geq j} p_i)/(m-j+1)\right\rceil\}\}$$

In what follows we will use a branch and bound method to solve the considered problem. For that, one defines the following procedures.

a. Evaluation procedure: To determine the evaluation function g one uses the result of the preceding theorem:

$g = \max\{\max_{1\leq j\leq m}\{\overline{p}_j\}, \max_{1\leq i\leq n}\{p_i\}, \max_{j\in E}\{\left\lceil (\sum_{i:h_i\geq j} p_i)/(m-j+1)\right\rceil\}\}$

where $\overline{p}_j$ is the sum of the processing times of the jobs assigned to the machine M_j. The evaluation procedure enables us to build a scheduling with length lower or equal to g. For that, one supposes that a list of priorities is given to the jobs not yet assigned as a LPT (Longest Processing Time) rule, and to each stage, the first available machine is selected to process the first free job of the list such as the total processing time, on the considered machine, be less or equal to g.

If a job J_i cannot be assigned to a machine M_j so that $\overline{p}_j + p_i \leq g$, then J_i is shared between the machines having an index $\geq h_i$, and the solution obtained is not feasible. The scheduling obtained is of length equal to g.

b. Separation procedure: On each of the nodes of the tree structure, define y as the best exact evaluation (associated to a feasible solution) found and g is the calculated evaluation on the considered node.

If g is larger than $y-1$, it is not necessary to explore more this node, since there cannot be an optimal solution of cost lower than $y-1$ starting from this node.

Else " $g \leq y-1$ " one looks whether the obtained solution associated to the evaluation g is feasible; if this is the case one assigns to y the value of g; if not, one separates as follows:

Let J_i be the job which caused the non feasibility of the solution. Starting from the considered node one will create $|m-h_i+1|$ nodes such that: J_i is assigned to M_{h_i} for the first node, J_i is assigned to M_{h_i+1} for the second node, ... , J_i is affected to M_m for the $|m-h_i+1|$th node. Let us note that we use a backtracking exploration of the tree structure.

The algorithm A4 is composed of three procedures:

Procedure initialization ;
begin - $M^*_1, \ldots, M^*_m := \emptyset$; {no job is assigned to the machines}
- $JC := J$; $y := +\infty$; $\overline{p}_1 := 0; \ldots; \overline{p}_m := 0$
end ;
Procedure evaluation(input $M^*_1, \ldots, M^*_m, JC$; input/output g ; output $S, bool$) ;
begin - $g = \max\{\max_{1\leq j\leq m}\{\overline{p}_j\}, \max_{1\leq i\leq n}\{p_i\}, \max_{j\in E}\{(\sum_{i:h_i\geq j} p_i)/(m-j+1)\}\}$;
- Assign the jobs of JC to the machines of M in the non increasing order of h_i
(and according to the non increasing order of the processing times in the case of equality) such that: $\overline{p}_j \leq g \; \forall j = 1, \ldots, m$;
- if a job J_i cannot be assigned to a machine M_j ($\overline{p}_j > g$)
then $S := i$; $bool := false$ else $bool := true$
endif
end ;
Procedure SE(input $M^*_1, \ldots, M^*_m, JC$; input/output y);
begin - evaluation $(M^*_1, \ldots, M^*_m, JC, g, S, bool)$;
- if $g < y$

then if $bool = true$
then y:=g ; $\overline{M}_1 := M_1^*, \ldots, \overline{M}_m := M_m^*$
else {separation}
- $j := h_S$; $JC := JC \setminus \{J_S\}$;
- while $(j \leq m)$ and $(g < y)$
do - $M_j^* := M_j^* \cup \{J_S\}$; $\overline{p}_j := \overline{p}_j + p_j$;
- $SE(M_1^*, \ldots, M_m^*, JC, y)$;
- $M_j^* := M_j^* \setminus \{J_S\}$; $\overline{p}_j := \overline{p}_j - p_j$; $j := j + 1$
enddo
endif
endif
end ;

This algorithm is of course finite.

5 Heuristics

The exponential exact methods are non efficient for the NP-hard problems, for this reason polynomial methods have appeared, which not necessarily give an optimal solution but seek by various techniques the best possible approaching solution.

Problem $PMPMO//C_{max}$

To solve the problem $PMPMO//C_{max}$ we will use the algorithm of the problem $PMPMO$ $/pmtn/C_{max}$ to develop an algorithm which gives an 2-approximation, i.e. a feasible scheduling S' such as $C_{max}^{S'} \leq 2.C_{max}^*$.

Algorithm A5 ;
begin - Solve the problem $PMPMO$ $/pmtn/C_{max}$ by using the algorithm A3;
- Reschedule preempted jobs on all machines while respecting feasibility of the solution
end.

Theorem 8 *The algorithm A5 gives an 2-approximation for the problem* $PMPMO//C_{max}$.

In the continuation of this part, we will propose an heuristic of resolution, which gives a result better than that of the algorithm A5, and in $O(nlogn + nm)$. To this end, we use an LPT arrangement by giving a certain order of processing of the jobs on the machines, according to the feasibility of the imposed constraints.

Algorithm A6 ;
begin while $E \neq \emptyset$
do - Choose the greatest index of E (let j be this index) ;
- $E := E \setminus \{j\}$; $F_j := \{J_i \in J / h_i = j\}$; $J := J \setminus \{Fj\}$;
- Arrange the jobs of F_j according to the rule LPT;
- for $k = 1$ to $|F_j|$
do Schedule the first job, not yet processed, of the set F_j,
on the first free machine among the machines $M_j, \ldots, M_m$.
enddo
enddo
end.

Theorem 9 *The heuristic A6 runs in* $O(nlogn + nm)$.

6 Numerical experimentations

Algorithms proposed in this paper (heuristics and exact method) for the problem $PMPM/\ /C_{max}$ were tested and compared on instances generated randomly according to an uniform law. Several numbers of machines ($m \in \{4, 6, 12, 20\}$) and jobs ($n \in \{7, 10, 15, 20, 25, 30, 35,\ 40, 50, 60, 80, 100\}$) were used for the various tests. Some numerical results obtained are given and summarized.

The various tests realized (on a pentium IV) have shown that for instances of small size, the optimal solution is found quickly in a reasonable time. The approached solution is to be very near the optimum with a ratio of performance lower than 1.2 in the majority of the cases and lower than 1.1 in more than 90 % of the cases. For the instances of big size, the exact method did not enable us to determine the optimal solution but the results obtained confirm the efficiency of the heuristics A6.

7 Conclusion

In this paper, we have shown that the problem of minimization of the makespan on ordered parallel machines is NP-hard. We have presented exact polynomial algorithms to solve some easy subproblems. Two heuristics and an exact method with numerical experiments are also presented. Let us note that the heuristic ones are generally used to tackle with this type of NP-hard problems.

References

[1] J. Blazewicz (1987), Selected topics in scheduling theory, *Annals of discrete mathematics* **31**, 1 – 60.

[2] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Weglarz (1996), *Scheduling computer and manufacturing processes*, Springer-Verlag, Berlin.

[3] J. Blazewicz, J. Lazewicz, K.H. Ecker, G. Schmidt and J. Weglarz (1994), *Scheduling in computer and manufacturing systems*, Springer-Lehrbuch, Berlin.

[4] M. Boudhar (1996), *Sur quelques problèmes d'ordonnancement d'atelier*, thèse de magister, USTHB-Algiers.

[5] P. Brucker (1995), *Scheduling algorithms*, Springer-Verlag, Berlin.

[6] K.R. Baker (1974), *Introduction to sequencing and scheduling*, John Wiley & Sons, New York.

[7] P. Brucker and S. Knust (2006), *Complexity results of scheduling problems*, page web: http://www.mathematik.uni-osnabrueck.de/research/OR/class/.

[8] J. Carlier and P. Chretienne (1988), *Problèmes d'ordonnancement : modélisation, complexité et algorithmes*, Masson, Paris.

[9] M.R. Garey and D.S. Johnson (1979), *Computers and intractability: a guide to the theory of NP-Completeness*, W.H. Freeman and Company, San Francisco.

[10] V. Zsuzsanna (2005), On scheduling problems with parallel multi-purpose machines, *EGRES Technical Report TR-2005-02*, Etvs University, Budapest.

A Parallel Metaheuristics for the Single Machine Total Weighted Tardiness Problem with Sequence-Dependent Setup Times

Wojciech Bożejko
Wrocław University of Technology, Institute of Computer Engineering, Control and Robotics, Janiszewskiego 11-17, 50-372 Wrocław, Poland, wojciech.bozejko@pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław
Institute of Computer Science, Joliot-Curie 15, 50-383 Wrocław, Poland, mwd@ii.uni.wroc.pl

In the paper we propose a parallel population-based metaheuristics for solving a single machine scheduling problem with total weighted tardiness criterion and sequence-dependent setup times. It is represented by $1|s_{ij}|\sum w_iT_i$ in literature and it belongs to the strongly NP-hard class. Calculations on the representative group of benchmark instances were done and results were compared with the best known from literature. Obtained solutions were better than the benchmark ones in many instances.

Keywords: Meta-heuristic Search.

1 Introduction

The single machine total weighted tardiness problem with sequence-dependent setup times is denoted in literature as $1|s_{ij}|\sum w_iT_i$ and it is strongly NP-hard, because $1||\sum w_iT_i$ (with $s_{ij} = 0$) is strongly NP-hard (see Lenstra, Rinnoy Kan and Brucker (1977)). To date the best construction heuristics for this problem has been the Apparent Tardiness Cost with Setups (ATCS – Lee, Bhaskaran and Pinedo (1997)). Many metaheuristics have also been proposed. Tan et al. (2000) presented a comparison of four methods of solving the considered problem: Branch and Bound, Genetic Search, random-start pair-wise interchange and Simulated Annealing. Gagné, Price and Gravel (2002) compared an Ant Colony Optimization algorithm with other heuristics. Cicirello and Smith (2005) proposed benchmarks for the single machine total tardiness problem with sequence-dependent setups by generated 120 instances and applied stochastic sampling approaches: Limited Discrepancy Search (LDS), Heuristic-Biased Stochastic Sampling (HBSS), Value Biased Stochastic Sampling (VBSS), Value Biased Stochastic Sampling seeded Hill-Climber (VBSS-HS) and Simulated Annealing. The best goal function value obtained by their approaches was published in literature and presented at http://www.ozone.ri.cmu.edu/benchmarks.html as the upper bounds of the benchmark problems. This upper bounds was next improved by Cicirello (2006) by Genetic Algorithm, Lin and Ying (2006) by Tabu Search, Simulated Annealing and Genetic Algorithm, and Liao and Juan (2007) by an Ant Optimization.

In this paper we propose a method by which we have obtained the new better upper bound values. It is based on the idea which we have introduced in the paper [1].

2 Definition of the problem

Let $N = \{1, 2, \ldots, n\}$ be a set of n jobs which have to be processed, without an interruption, on one machine. This machine can process at the most one job in any time. For a job i ($i = 1, 2, \ldots, n$), let p_i,w_i,d_i be: a *time of executing*, a *weight of the cost function* and a *deadline*. Let s_{ij} be a setup

time which represents a time is needed to prepare the machine for executing a job j after finishing executing a job i. Additionally s_{0i} is a time which is needed to prepare a machine for executing the first job i (at the beginning of the machine work). If a sequence of job's executing is determined and C_i (i =1,2, ..., n) is a time of finishing executing a job i, then T_i= $\max\{0, C_i - d_i\}$ we call a *tardiness*, and $f_i(C_i) = w_i T_i$ *a cost of tardiness* of a job i. The considered problem consists in determining such a sequence of executing of jobs which minimizes a *sum of costs of tardiness*, i.e. $\sum w_i T_i$.

Let Π be a set of permutations of elements from the set N. For a permutation $\pi \in \Pi$ by

$$F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)}),$$

we represent a *cost of permutation* π (i.e. a sum of costs of tardiness, when jobs are executed in a sequence determined by a permutation π), where $C_{\pi(i)} = \sum_{j=1}^{i} (s_{\pi(j-1)\pi(j)} + p_{\pi(j)})$ and $\pi(0) = 0$. The considered problem consists in determining a permutation $\pi \in \Pi$ which has a minimal sum of costs of tardiness.

3 Parallel Population-Based Metaheuristics

We present a method belonging to the population-based approaches which consists in determining and researching the local minima. This (heuristic) method is based on the following observation. If there are the same elements in some positions in several solutions, which are local minima, then these elements can be in the same position in the optimal solution.

Since we propose this method for solving problems in which a solution is a permutation, so in the next part of the paper we identify these two terms.

The basic idea is to start with an initial population (any subset of the solution space). Next, for each element of the population, a local optimization algorithm is applied (e.g. descending search algorithm or a metaheuristics) to determine a local minimum. In this way we obtain a set of permutations – local minima. If there is an element which is in the same position in several permutations, then it is fixed in this position in the permutation, and other positions and elements of permutations are still free. A new population (a set of permutations) is generated by drawing free elements in free positions (because there are fixed elements in fixed positions). After determining a set of local minima (for the new population) we can increase the number of fixed elements. To prevent from finishing the algorithm's work after executing some number of iterations (when all positions are fixed and there is nothing left to draw), in each iteration "the oldest" fixed elements are set as free.

For the parallel version of the algorithm a global model of parallelization has been proposed. This model executes multiple population training metaheuristics synchronizing populations in each iteration, i.e. common global table of the fixed elements and positions is used for each processor. In every iteration the average number $nr(a, l)$ of permutations (for all subpopulations) in which there is an element a in a position l is computed.

Let Π be a set of all permutations of elements from the set $N = \{1, 2, \ldots, n\}$ and the function:

$$F : \Pi \to R^+ \cup \{0\}.$$

We consider a problem which consists in determining optimal permutation $\pi_{opt} \in \Pi$. We use the following notation: π^* – sub-optimal permutation determined by the algorithm, η – number of elements in the population, P^i – population in the iteration i of the algorithm, $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$, $LocalOpt(\pi)$ – local optimization procedure to determine local minimum where π is a starting

solution, LM^i – a set of local minima in iteration i, $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ ... \ , \hat{\pi}_\eta\}$, $\hat{\pi}_j = LocalOpt(\pi_j)$, $\pi_j \in P^i$, $j = 1, 2, \ ... \ , \eta$, FS^i – a set of fixed elements and position in permutations of population P^i, $FixSet(LM^i, FS^i)$ – a procedure which determines a set of fixed elements and positions in the next iteration of the population-based algorithm, $FS^{i+1} = FixSet(LM^i, FS^i)$, $NewPopul(FS^i)$ – a procedure which generates a new population in the next iteration of algorithm, $P^{i+1} = NewPopul(FS^i)$.

In any permutation $\pi \in P^i$ positions and elements which belong to the set FS^i (in iteration i) we call *fixed*; other elements and positions we call *free*.

The algorithm begins by creating an initial population P^0 (and it can be created randomly). We set a sub-optimal solution π^* as the best element of the population P^0,

$$F(\pi^*) = min\{F(\beta) : \ \beta \in P^0\}.$$

A new population of iteration $i + 1$ (a set P^{i+1}) is generated as follows: for a current population P^{i+1} a set of local minima LM^i is determined (for each element $\pi \in P^i$ executing procedure $LocalOpt(\pi)$). Elements which are in the same positions in local minima are established (procedure $FixSet(LM^i, FS^i)$), and a set of fixed elements and positions FS^{i+1} is generated. Each permutation of the new population P^{i+1} contains the fixed elements (in fixed positions) from the set FS^{i+1}. Free elements are randomly drawn in the remaining free positions of permutation.
If permutation $\beta \in LM^i$ exists and $F(\beta) < F(\pi^*)$, then we update π^* ($\pi^* \leftarrow \beta$). The algorithm finishes (*Stop Criterion*) after executing the *Max_iter* iterations.

The general structure of the parallel population-based heuristic algorithm for the permutation optimization problem for each processor is given below. The $FixSet$ and $NewPopul$ algorithms are described in the further part of the paper.

Algorithm 1. Parallel Population-Based Metaheuristics (ParPBM)
Initialization:
$P^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$; *starting populations for each of nrtasks populations*
$\pi^* \leftarrow$ the best element of the population P^0;
$i \leftarrow 0$;
$FS^0 \leftarrow \emptyset$;
parfor $p = 1..nrtasks$ **do**
  repeat
    For each process p determine sets of local minima
      $LM_p^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where $\hat{\pi}_j \leftarrow LocalOpt(\pi_j), \pi_j \in P_p^i$;
    for $j \leftarrow 1$ **to** η **do** **if** $F(\hat{\pi}_j) < F(\pi^*)$ **then** $\pi^* \leftarrow \hat{\pi}_j$;
    Determine sets $FS^{i+1} \leftarrow FixSet(LM^i, FS^i)$;
    Generate new populations $P^{i+1} \leftarrow NewPopul(FS^i)$;
    $i \leftarrow i + 1$;
  until $i > max\_iter$;
end parfor.

Local optimization (*LocalOpt* procedure.) A fast method based on the local improvement is applied to determine the local minima. The method begins with an initial solution π^0. In each iteration for the current solution π^i the neighborhood $\mathcal{N}(\pi^i)$ is determined. Next, from the neighborhood it is chosen the best element π^{i+1} is chosen which is the current solution in the next iteration.

In the implementation of the *LocalOpt* procedure a very quick *descent search* algorithm is applied. The neighborhood is generated by swap moves (s-moves). Local search procedure does not preserve the positions of fixed elements to obtain better (closer to optimal) values of the local minima.

A set of fixed elements and position (*FixSet* procedure). The set FS^i (in iteration i) includes quadruples (a, l, α, φ), where a is an element of the set $N = \{1, 2, \ldots, n\}$, l is a position in the permutation ($1 \leq l \leq n$) and α, φ are attributes of a pair (a, l). A parameter α means "*adaptation*" and decides on inserting to the set, and φ – "*age*" – decides on deleting from the set. Parameter φ enables to set free a fixed element after making a number of iterations of the algorithm. However, a parameter α determines a fraction of local minima, in which an element a is in position l.
Both of these parameters are described in a further part of this chapter. The maximal number of elements in the set FS^i is n. If the quadruple (a, l, α, φ) belongs to the set FS^i, then there is an element a in the position l in each permutation from the population P^i.

In each iteration of the algorithm, after determining local minima (*LocalOpt* procedure), a new set $FS^{i+1} = FS^i$ is established. Next, a $FixSet(LM^i, FS^i)$ procedure is invoked in which the following operations are executed:

(1) fixing the new elements,

(2) erasing the oldest,

(3) modifying of the age of each element.

There are two functions of acceptance Φ and Γ connected with the operations of inserting and deleting. Function Φ is determined by an auto-tune function. Function Γ is fixed experimentally as a constant.

Fixing elements. Let $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$ be a population of η elements in the iteration i. For each permutation $\pi_j \in P^i$, applying the local search algorithm (*LocalOpt*(π_j) procedure), a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$ is determined. For any permutation $\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), \ \ldots, \hat{\pi}_j(n)), \quad j = 1, 2, \ \ldots \eta$, let be $nr(a, l) = \left|\{\hat{\pi}_j \in LM^i : \quad \hat{\pi}_j(l) = a\}\right|$. It is a number of permutations from the set LM^i in which the element a is in the position l.
If $a \in N$ is a free element (i.e. $a \notin FS^i$) and $\alpha = \frac{nr(a,l)}{\eta} \geq \Phi(i)$, then the element a is fixed in the position l; we assume $\varphi = 1$ and the quadruple (a, l, α, φ) is inserted to the set of fixed elements and positions, it means $FS^{i+1} \leftarrow FS^i \cup \{(a, l, \alpha, \varphi)\}$.

Auto-tune of the acceptance level Φ. Acceptance function Φ is defined so that

$$\forall i, \ \ 0 < \Phi(i) \leq 1.$$

It is possible that no element is acceptable to be fixed in an iteration, or too many elements (i.e. a half) are fixed in one iteration. To prevent from it an auto-tune procedure for Φ value is proposed. In each iteration i, if

$$\max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} < \Phi(i) \quad \text{or} \quad \max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} \gg \Phi(i)$$

therefore, $\Phi(i)$ value is fixed as

$$\Phi(i) \leftarrow \max_{a,l \in \{1,2,\ldots,n\}} \frac{nr(a,l)}{\eta} - \epsilon,$$

where ϵ is a small constant, e.g. $\epsilon = 0.05$. In this way the value of $\Phi(i)$ is adopted to the problem instance specificity. In this implementation of the algorithm $\Phi(0) = 0.7$ as a starting value.

Deleting elements. Each fixed element is released after executing some number of iterations to makes possible testing a plenty of local minima. In this implementation function $\Gamma(i)$ is defined as a constant equals 2, so each element of the set FS^i is deleted after executing 2 iterations.

In the i-th iteration of the ParPBM algorithm a set of fixed elements and positions FS^{i+1} is determined by using the following algorithm.

Algorithm 2. FixSet(LM^i, FS^i)
Input: a set of local minima LM^i;
Output: a set of fixed elements and position FS^i;
for all $k, j \in N$ **do** $nr(k,j) \leftarrow 0$; **end for**;
for j:=1 **to** η **do**
for a:=1 **to** n **do**
$nr(a, \hat{\pi}_j(a)) \leftarrow nr(a, \hat{\pi}_j(a)) + 1$;
end for;
end for;
for a:=1 **to** n **do**
if (a is a free job) **then**
choose l such, that $nr(a,l) = \max_{1 \le s \le n} \{nr(a,s)\}$;
if $nr(a,l) \ge \Phi(i)$ **then**
fix an element a on the position l
(i.e. $FS^{i+1} \leftarrow FS^i \cup \{(a,l,\alpha,\varphi)\}$, where $\alpha = \frac{nr(a,l)}{\eta}$ and $\varphi = 0$;)
end for;
{*Increasing age of each fixed element and setting free the old ones*}
for a:=1 **to** n **do**
$\varphi(a) \leftarrow \varphi(a) + 1$;
if $\varphi(a) > \Gamma(i)$ **then** set free an element a
(i.e. $FS^{i+1} \leftarrow FS^{i+1} \setminus \{(a,l,\alpha,\varphi)\}$);
end for.

Procedure *NewPopul*. Let a quadruple $(a,l,\alpha,\varphi) \in FS^{i+1}$. Therefore, in each permutation of a new population P^{i+1} there exists an element a in a position l. Randomly drawn free elements will be inserted in remaining (free) positions. A function *random* generates an element of the set FE from the uniform distribution. Computational complexity of the *NewPopul* algorithm is linear. Population P^{i+1} is generated as follows:

Algorithm 3. New Population (*NewPopul*(FS_{i+1}))
$P^{i+1} \leftarrow \emptyset$;
Determine a set of free elements:
$FE \leftarrow \{a \in N : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$
and a set of free positions:
$FP \leftarrow \{l : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$;
for $j \leftarrow$ **to** η **do**
for every $(a, l, \alpha, \varphi) \in FS^{i+1}$ **do** $\pi_j(l) \leftarrow a$; **end for**;
for $s \leftarrow 1$ **to** n **do**
if $s \in FP$ **then**
$w \leftarrow random(FE)$ **and** $FE \leftarrow FE \backslash \{w\}$;
$\pi_j(s) \leftarrow w$;
end for;
$P_{i+1} \leftarrow P_{i+1} \cup \{\pi_j\}$.
end for.

4 Computational experiments

Parallel population-based metaheuristics was implemented in C++ language with the MPI library (MPICH) and it was tested on the cluster of 8 computers with Intel Celeron 2.4GHz processors and 240MB RAM memory working under Windows 2000 operating system connected by Ethernet 100mbit/s. Computational experiments were done to compare the obtained results with the benchmarks from literature (Cicirello and Smith (2005), represented by a in the tables) and the newest obtained results for this single machine problem (Liao and Juan (2007), ACO, represented by b; Lin and Ying (2006): SA, GA, TS, represented by c, d, e, respectively; and SA-TABU, f (on line).

The following tuning parameters was used: $max\_iter = 520$, threshold $\epsilon = 0.05$ for Φ adjusting, population size per each processor $\eta = 200$, maximal age $\Gamma(i) = 2$.

In the Table 1 results of computational experiments for the problem $1|s_{ij}|\sum w_i T_i$ are presented with the new upper bounds marked. Average percentage deviation δ of the obtained solution value F_{ParPBM} to benchmark ones F_{bench} from Cicirello and Smith (2005) is also computed: $\delta = \frac{F_{ParPBM} - F_{bench}}{F_{bench}} \cdot 100\%$. Benchmark values F_{bench} are not the best knows, but all the authors are comparing to them, so the average value of δ is rational measurement of the algorithm efficiency. As we can see it was possible to find 18 new upper bounds of the optimal cost function for the benchmark instances. The time consumed by all processors took 7 minutes in average.

Table 2 presents the average percentage deviation from the benchmark solutions of the newest approaches to the concerned problem compared to sequential implementation of the ParPBM (for one processor). It was on the level of $-10,05\%$ with average time 20 seconds and was better than earlier proposed approaches to this problem with comparable times of execution (the authors of SA^c, GA^d and TS^e announce 27 seconds in average on Pentium IV 1.4GHz).

5 Conclusion

We have discussed a new approach to the permutation optimization problems based on the parallel population-based metaheuristic algorithm. Usage of the population with fixed features of local optima makes the performance of the method much better than the iterative improvement approaches, such as in tabu search, simulated annealing as well as classical genetic algorithms.

Table 1: Results of computational experiments for the problem $1|s_{ij}|\sum w_i T_i$. The new upper bounds are marked by bold font.

Nr	F_{best}	F_{ParPBM}	$\delta(\%)$	Nr	F_{best}	F_{ParPBM}	$\delta(\%)$
1	684^{d}	696	-28,83%	61	76357^{f}	76373	-4,40%
2	5082^{e}	5367	-17,29%	62	44769^{f}	44869	-6,25%
3	1792^{c}	**1782**	-24,11%	63	75317^{c}	76146	-3,39%
4	6526^{d}	6615	-20,41%	64	92572^{c}	92860	-3,65%
5	4662^{d}	4774	-14,84%	65	126907^{f}	128593	-4,66%
6	5788^{b}	7500	-9,02%	66	59717^{f}	59852	-6,56%
7	3693^{d}	3765	-13,39%	67	29390^{d}	29394	-15,77%
8	142^{d}	153	-53,21%	68	22148^{e}	22528	-14,68%
9	6264^{f}	6628	-12,77%	69	64632^{b}	71534	-5,14%
10	2021^{c}	2099	-14,36%	70	75102^{d}	75801	-6,65%
11	3867^{d}	4452	-15,41%	71	150053^{f}	151866	-5,81%
12	0^{a}	0	0,00%	72	46903^{e}	50171	-11,88%
13	5685^{d}	5797	-5,69%	73	29408^{c}	30529	-16,28%
14	3045^{d}	3205	-18,68%	74	33375^{e}	34357	-10,28%
15	1458^{d}	1788	-38,66%	75	21863^{e}	23244	-24,97%
16	4940^{d}	**4939**	-26,40%	76	55055^{c}	56198	-16,81%
17	204^{c}	**194**	-58,01%	77	33303^{f}	35932	-11,41%
18	1610^{d}	1723	-31,46%	78	20632^{f}	22256	-11,35%
19	36^{f}	**0**	-100,00%	79	119099^{f}	121085	-3,77%
20	2967^{d}	3273	-21,94%	80	20161^{f}	23138	-27,34%
21	0^{a}	0	0,00%	81	384996^{d}	385528	-0,42%
22	0^{a}	0	0,00%	82	410458^{f}	**410432**	-0,74%
23	0^{a}	0	0,00%	83	459817^{f}	460459	-1,20%
24	1063^{d}	**1060**	-40,82%	84	330384^{c}	330837	-0,25%
25	0^{a}	0	0,00%	85	555106^{c}	556891	-0,30%
26	0^{a}	0	0,00%	86	364075^{f}	365924	0,04%
27	0^{d}	0	-100,00%	87	399214^{f}	401482	-0,38%
28	0^{d}	0	-100,00%	88	434948^{d}	436394	-0,11%
29	0^{a}	0	0,00%	89	410966^{c}	**410909**	-1,44%
30	165^{c}	**159**	-72,35%	90	402233^{d}	403601	-0,82%
31	0^{a}	0	0,00%	91	344988^{e}	346111	-0,31%
32	0^{a}	0	0,00%	92	364593^{f}	**362823**	-0,81%
33	0^{a}	0	0,00%	93	410462^{a}	413534	0,75%
34	0^{a}	0	0,00%	94	335210^{f}	335542	-0,23%
35	0^{a}	0	0,00%	95	519364^{f}	523309	-0,87%
36	0^{a}	0	0,00%	96	461484^{b}	462961	-0,31%
37	755^{c}	986	-59,04%	97	413109^{e}	417890	-0,57%
38	0^{a}	0	0,00%	98	532519^{a}	530958	-0,29%
39	0^{a}	0	0,00%	99	370084^{b}	373734	-0,28%
40	0^{a}	0	0,00%	100	439944^{d}	**437403**	-1,01%
41	71186^{e}	71418	-2,40%	101	353408^{e}	**353141**	-0,75%
42	58199^{e}	59377	-4,01%	102	493889^{e}	494300	-0,37%
43	147211^{c}	147721	-1,51%	103	379913^{b}	**379195**	-0,26%
44	35648^{c}	35742	-7,71%	104	358334^{c}	358741	-0,90%
45	59307^{d}	59572	-5,08%	105	450808^{c}	**450806**	-1,22%
46	35320^{e}	35909	-5,48%	106	455849^{d}	457420	-0,54%
47	73972^{f}	74024	-4,10%	107	352855^{f}	**352821**	-1,07%
48	65164^{c}	65943	-4,32%	108	462737^{e}	463753	-0,93%
49	79055^{e}	79224	-5,85%	109	413205^{c}	413569	-0,54%
50	32743^{f}	33201	-8,37%	110	419481^{e}	420152	-0,27%
51	52163^{f}	**51854**	-11,47%	111	347233^{b}	349805	-0,26%
52	99200^{d}	102688	-2,54%	112	373238^{b}	376036	-0,37%
53	91302^{c}	92111	-3,50%	113	261239^{d}	**261001**	-0,84%
54	122968^{f}	124708	0,93%	114	470327^{b}	472780	-0,09%
55	69571^{f}	71657	-6,17%	115	459194^{b}	464415	0,91%
56	78960^{d}	79756	-9,80%	116	527459^{b}	537799	-0,45%
57	69320^{c}	**68489**	-2,73%	117	512028^{f}	**508415**	-1,96%
58	48081^{e}	48637	-12,40%	118	352118^{b}	357087	-0,14%
59	55396^{c}	56376	-4,54%	119	579462^{e}	584046	0,02%
60	68176^{f}	69103	-5,76%	120	398590^{b}	402422	0,68%
				Average			**-10,28**

Table 2: Average improvement rates (%) of the SA, GA and TS compared to the proposed ParPBM (sequential version) approach. Standard deviation of the ParPBM results σ^{PHM} over 10 runs.

Problem set	SA[c]	GA[d]	TS[e]	ParPBM	σ^{ParPBM}
1 to 10	20.00	22.83	19.12	20.82	3.47
11 to 20	20.89	27.60	18.46	31.63	33.62
21 to 30	30.39	30.93	29.18	29.79	31.85
31 to 40	6.86	6.42	5.81	5.90	3.59
41 to 50	5.21	5.65	5.33	4.83	2.02
51 to 60	5.29	5.65	4.44	5.01	6.11
61 to 70	7.25	6.56	7.25	7.06	3.80
71 to 80	15.39	15.02	16.32	13.89	5.80
81 to 90	0.66	0.56	0.56	0.46	0.40
91 to 100	-0.47	-0.50	-0.11	0.37	0.64
101 to 110	0.60	0.24	0.64	0.68	0.45
111 to 120	-0.23	-0.44	-0.23	0.19	1.01
Average	9.32	9.97	8.90	10.05	7.73

References

[1] W. Bożejko and M. Wodecki (2006), Evolutionary Heuristics for Hard Permutational Optimization Problems, *International Journal of Computational Intelligence Research* **2**(2), Research India Publications, 151 – 158.

[2] V.A. Cicirello and S.F. Smith (2005), Enhancing stochastic search performance by value-based randomization of heuristics, *Journal of Heuristics* **11**, 5 – 34.

[3] V.A. Cicirello (2006), Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respect Absolute Position, *8th Annual Genetic and Evolutionary Computation Conference GECCO 2006*, ACM Press, 1125 – 1131.

[4] C. Gagné, W.L. Price and M. Gravel (2002), Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times, *Journal of the Operational Research Society* **53**, 895 – 906.

[5] Y.H. Lee, K. Bhaskaran and M. Pinedo (1997), A heuristic to minimize the total weighted tardiness with sequence-dependent setups, *IIE Transactions* **29**, 45 – 52.

[6] J.K Lenstra, A.G.H. Rinnoy Kan and P. Brucker (1977), Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics* **1**, 343 – 362.

[7] C.-J. Liao and H. C. Juan (2007), An ant opimization for single-machine tardiness shceduling with sequence-dependent setups, *Computers & Operations Research* **34**, 1899 – 1909.

[8] S.-W. Lin and K.-C. Ying (2006), Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics, *International Journal of Advanced Manufacturing Technology* (on line), DOI 10.1007/s00170-006-0693-1.

[9] S.-W. Lin and K.-C. Ying, Hybrid simulated annealing and tabu search for solving single-machine tardiness problems with sequence-dependent setup times, (on line) http://pc33.mis.hfu.edu.tw/smswtsds/data/

[10] K.C. Tan, R. Narasimban, P.A. Rubin and G.L. Ragatz. (2000). A comparison on four methods for minimizing total tardiness on a single procesor with sequence dependent setup times, *Omega* **28**, 313 – 326.

Optimization of the Departure Schedule at a Public Transit Terminal with Multiple Destinations

Giuseppe Bruno, Gennaro Improta, Antonino Sgalambro
Universitá di Napoli Federico II, Dipartimento di Ingegneria Economico Gestionale
Piazzale Tecchio, 80 - 80125 Napoli, Italy, {giuseppe.bruno, gennaro.improta, antonino.sgalambro}@unina.it

The paper considers the schedule optimization problem for public transit networks. In particular, we are interested in optimizing the departure schedule for the lines that leave from a transit terminal, in which passengers are supposed to arrive, according to a given schedule, and split between different lines of a service, or even change mode of transportation in case of intermodal systems. The aim is to decide the schedule for the output lines, in such a way to find the optimal trade-off between the sum of service operative costs and passengers waiting times at the transit terminal. We present a model that is able to represent the problem in presence of a single destination for all the passengers, and its extension to the general case of multiple destination. We show similarities between this models and those proposed in literature for other well known combinatorial optimization problems, also drawing some conclusions about the complexity of the proposed models.

Keywords: Location, Logistics, Schedule Synchronization, Transit Network, Transport Scheduling.

1 Introduction

Intermodal transportation is based on the combined use of different modes of transport to move passengers from their own origins to destinations. The efficiency of such systems depends on the possibility to ensure transfers involving switching passengers from one route to another with limited waiting times. A crucial role in the efficiency of an intermodal system is played by transfer or transit terminals, i.e. locations where users can split line and/or modes of transport during their trips. The central issue is often the maximal synchronization of the schedule, given a set of transit lines crossing each other in some point of the networks, in order to minimize passenger waiting times. Such a problem is widely treated in literature as the Schedule Synchronization Problem (SSP). Klemt and Stemme [14] and Domschke [9] considered the problem of schedule synchronization for public transit networks with the objective of minimizing the sum of users transfer waiting times assuming given operation hours. Keudel [13] proposed an interactive optimization tool aiming at reducing the total users waiting times and the number of users exceeding a given time limit. Desilets and Rousseau [7] presented some results for the application of a variant of the SSP on the network of Montreal. In [21], Voss formulated the SSP as a multicommodity network design problem, exploiting the quadratic semi-assignment problem, and proposed a tabu search algorithm to solve the problem. Daduna and Voss [6] provided the application of the tabu search algorithm for the SSP on some case studies of practical interest. In [1], Adamski formulated and solved the problem of synchronizing different transit lines with shared route segments. Ceder et al. [3, 4] addressed the problem of generating the timetable for a given network of buses so as to minimize their synchronization. Wong and Leung proposed in [20] a mixed integer programming formulation aiming at the reduction of the waiting times for the passengers of a railway system, also proposing a heuristic approach. Schroder and Solchenbach [17] proposed a quadratic semi-assignment model based on offset decision variables and a system of assignment of qualitative penalties to the solutions. Most of the proposed models are in general characterized by a complexity such that the size of the

instances solvable at the optimum within reasonable computational times is limited and often not compatible with the dimension of real applications. In this paper we are interested in the dimensioning of the public transit service, deciding how many output lines should be activated, given certain levels of demand, in such a way to balance operative costs and user costs. In the following, we refer to this as the Schedule Optimization Problem (SOP). Hence, we propose a mathematical model based on a time-space representation of the network to describe this problem, and possible extensions in order to take into account the presence of different hypothesis. The similarity with other known combinatorial problems makes it possible to deduce some conclusions about the complexity of the proposed models.
The paper is organized as follows. In Section 2 the general description of the SOP is defined and illustrated. In Section 3 a mathematical model for the single destination SOP is proposed with the indication of the computational complexity. In Section 4 the extension to the multiple destination case is provided. Finally, some conclusions are presented.

2 The schedule optimization problem for a transit terminal

A transfer or transit terminal is a location, within an intermodal transportation system, where users can split lines and/or modes of transportation during their own origin-destination trips. Within a time horizon T, we consider the presence of a given set of input lines I, i.e. transit lines starting from a set of origins and arriving at the transit terminal, and a set of output lines O, i.e. transit lines that start from the terminal towards a set of given destinations. We associate to each input line:

- the arrival time at the transit terminal;
- a line-destination demand representing the number of passengers yielded by this line and directed to each destination;

Each of the output lines is characterized by:

- a destination;
- the time of departure from the transit terminal;
- a capacity, i.e. the maximum number of passengers that can be simultaneously transported on the line;
- an activation cost, paid by the service operator to activate an output line.

A passenger, during his origin-destination trip, will choose an input line $i \in I$ and will arrive at time a_i at the terminal; afterwards he will choose the first available output line $k \in O$ to reach the destination. Being $b_k \in T$ the departure time of such line from the terminal, each passenger using this trip will have a waiting time equal to $b_k - a_i$. The Schedule Optimization Problem (SOP) consists in determining the number of lines to be activated, and the departure time for each of these lines, in order to optimize a certain performance index, satisfying at the same time all the transport demand present at the transit terminal. We assume, as a performance index to be minimized, the weighted sum of the total waiting times for the users (user cost) and of the total activation costs of the lines (operator cost). An appropriate calibration of the weights used in the objective function permits to find the desired tradeoff between user and operator costs.

We propose a formulation for the SOP based on a discrete time expansion of the terminal node. The time horizon T is divided into n time periods, obtained by dividing T by the length of a time

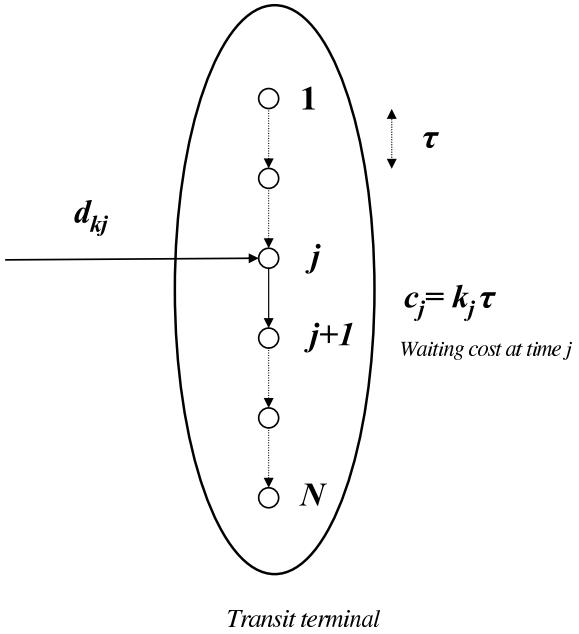


Figure 1: Sketch of the time-expanded transit terminal node: each holdover arc $(j, j+1)$ is associated with a waiting cost c_j.

unit τ. Therefore, the time expansion of the terminal node over the time horizon T is given by the graph $G_T = (N_T; A_T)$, being:

- $N_T = \{1, ..., n\}$
- $A_T = \{(j, j+1) : \forall j \in \{1, ..., n-1\}\}$

In this way, each node j in the time-expanded terminal node corresponds to a time instant of the time horizon T, that can coincide with the arrival and/or departure time of some line. Each node j is linked only to the successive node $j+1$; we refer to each arc $(j, j+1) \in A_T$ as holdover arc. To each holdover arc $(j, j+1) \in A_T$ is associated the waiting time τ. We can now define the waiting cost $c_j = k_j \cdot \tau$, with k_j representing a set of possible weights associated to the waiting time at time j. To each node $j \in N_T$ is associated a set of weights d_{kj} representing the total number of passengers that are supposed to arrive at the terminal at time j, and directed to the destination k. Figure 1 illustrates the discrete time expansion of the terminal node.

3 A mathematical model for the single destination problem

In the single destination SOP we suppose that the output lines are directed to a single common destination. Hence, it is possible to formulate the single destination SOP by defining the following sets of variables (see also Figure 2):

- x_j, $j \in N_T$, as continuous variables indicating the number of passengers which remain in the terminal during the time interval $[j, j+1]$, waiting for an available departure line;
- q_j, $j \in N_T$ as the number of passengers leaving the terminal at time j toward the destination using an output line activated at time j;

- y_j, $j \in N_T$ as binary variables assuming value 1 only if an output line is activated at time j.

In this way, the model is given by:

$$\min \quad \sum_{j \in N_T} c_j \cdot x_j + \sum_{j \in N_T} f_j \cdot y_j \tag{1}$$

$$\text{s.t.} \quad x_j = x_{j-1} + d_j - q_j \qquad \forall j \in N_T \tag{2}$$

$$q_j \leq M \cdot y_j \qquad \forall j \in N_T \tag{3}$$

$$x_1 = x_{n+1} = 0 \tag{4}$$

$$x_j \geq 0, \quad q_j \geq 0, \quad y_j \in \{0, 1\}$$

being:

- d_j the sum of passengers arriving at the terminal at time j and directed to the common destination;
- c_j the waiting cost associated to a passenger waiting at the transit terminal during interval $[j, j+1]$;
- f_j the fixed cost paid by the operator for the activation of a line starting at time j from the terminal.

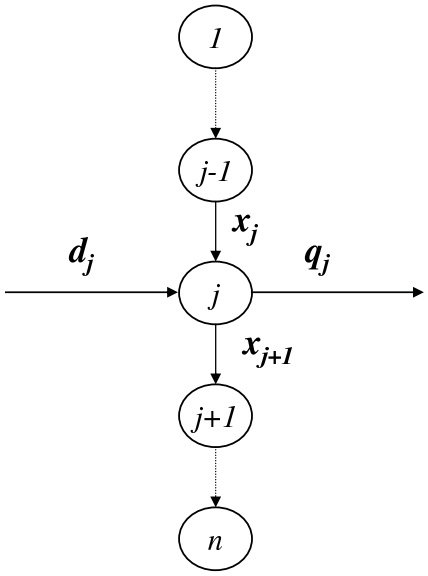


Figure 2: A scheme of the set of variables in the formulation.

This model uses $2 \cdot n$ continuous variables and n binary variables. The objective function (1) consists of the sum of the total users waiting costs and the total activation costs. Constraints (2) express the conservation of passenger flows at each time instant, while conditions (4) ensure that the number of passengers which stay in the terminal at the beginning and at the end of the time horizon is equal to 0. Constraints (3) ensure that variable q_j is equal to 0 if there is no output line starting at time j ($y_j = 0$). If we suppose that each of the activated output lines is large enough to

contain all the passengers that, at the time of the line departure, are waiting in the transit terminal (assuming that $M \geq \sum_{j=1}^{i} d_j$, $\forall i \in N_T$ in constraints (3), we refer to this case as the uncapacitated single destination SOP, otherwise we can consider a capacity upper bound Q_j for the number of passengers on a transit line which departs at time j, and replace constraints (3) by

$$q_j \leq Q_j \cdot y_j \qquad \forall j \in N_T \tag{5}$$

In this case we refer to the model as the capacitated single destination SOP.

3.1 Complexity of this model

In order to define the complexity we show that the model becomes perfectly equivalent to a very common formulation of the well known single-item Lot Sizing Problem (see for instance [8, 12, 19]). If we slightly modify the model 1-4 by replacing d_j with $-d'_j$ and q_j with $-q'_j$, we can view d'_j as the item quantity, or demand, that must be ready in a manufacturing system at time j, q'_j as the quantity of items to be produced at each time period, and x_j as the inventory of items at time j. In particular, it is possible to show that the uncapacitated single destination SOP falls in the class of polynomially solvable problems. This can be proved, for instance, by recalling and adapting the proof for the dynamical programming algorithm proposed in [18] by Wagner and Within. To do that, it suffices to prove that the following property holds:

(i) There exists an optimal solution with $x_j \cdot q_j = 0$, $\forall j \in N_T$ (whenever a line is activated, all the passengers get on board, and the terminal remains empty).

(ii) There exists an optimal solution in which, if $x_j > 0$, then $x_j = D_{kj}$, $k \leq j$, being $D_{kj} = \sum_{i=k}^{j} d_i$ (if someone leaves from the terminal at time j, the amount of passengers leaving at time j is equal to the entire demand of the $k+1$ consecutive periods from $j-k$ to j).

Proof. Let x_j and q_j be concurrently strictly positive for some $j \in N_T$. It would be feasible to increase q_j by x_j units and set x_j to zero, reducing the value of the objective function by $c_j \cdot x_j$. Therefore, at the optimum $x_j \cdot q_j = 0$, $\forall j \in N_T$. The second statement follows immediately.

The introduction of capacity constraints (5) makes the problem NP-hard. Indeed, it now corresponds to the Capacitated Lot Sizing model, that was proved in [10] to be NP-hard by reduction from the Knapsack Problem.

4 A mathematical model for the multi-destination problem

The extension of model (1)-(4) to the more general case with multiple destinations can be performed through a modification of the notation. The presence of a set K of possible destinations, can be taken into account by introducing the following variables:

- x_{kj}, $k \in K$, $j \in N_T$, as continuous variables representing the number of passengers which remain in the terminal during the time interval $[j, j+1]$, being directed to destination k;
- q_{kj}, $k \in K$, $j \in N_T$, as the number of passengers leaving the terminal at time j toward the destination k;
- y_{kj}, $k \in K$, $j \in N_T$, as binary variables which indicate whether an output line is activated at time j with destination k.

A scheme of this notation is sketched in Figure 3. The model for the multi-destination SOP can now be stated as follows:

$$\min \quad \sum_{k\in K}\sum_{j\in N_T}(c_j \cdot x_{kj} + f_j \cdot y_{kj}) \tag{6}$$

$$\text{s.t.} \quad x_{kj} = x_{k,j-1} + d_{kj} - q_{kj} \qquad \forall k \in K, \forall j \in N_T \tag{7}$$

$$q_{kj} \leq Q \cdot y_{kj} \qquad \forall k \in K, \forall j \in N_T \tag{8}$$

$$\sum_{k\in K} q_{kj} \leq C_j \qquad \forall j \in N_T \tag{9}$$

$$x_{k1} = x_{k,n+1} = 0 \qquad \forall k \in K \tag{10}$$

$$x_{kj} \geq 0, \quad q_{kj} \geq 0, \quad y_{kj} \in \{0,1\}$$

with c_j and Q having the same meaning of the single destination problem, and C_j representing the maximum number of passengers leaving the terminal at time j due to the capacity of the terminal. The objective (6) holds the same meaning of the single destination case, and represents the sum of total users waiting costs and the total activation costs. Constraints (7) represent the conservation of flow passengers directed to the same destination k at each time j. Constraints (8) are the capacity constraints associated to the output lines, that guarantee at the same time the activation of the related binary variable. Restrictions (9) limit the total number of passengers leaving the terminal at time j, while constraints (10) ensure that the number of passengers in the terminal at the beginning and at the end of the time horizon is equal to 0.

4.1 Complexity of the models

Adopting the same procedure shown for the single destination case, if we replace d_{kj} with $-d'_{kj}$, and q_{kj} with $-q'_{kj}$, we can interpret d'_{kj} as the external demand for an item k at time j, and q'_{kj} as the production quantity for item k at time j. In this way, the model (6)-(10) can be viewed as the well known multi-item Capacitated Lot Sizing problem which is known to be NP-Hard [2],[10].

In absence of restrictions (9) it would be possible to decompose the multi-destination problem into $|K|$ independent capacitated single destination problems, which we showed to be NP-Hard.

5 Conclusion

In this paper we defined the Schedule Optimization Problem at a transit terminal which consists of the determining the number of the lines and the departure times for each of these lines, directed to a set of possible destinations, in order to minimize the sum of the total users waiting costs and of the total activation costs of the lines. For this problem we provided some integer linear mathematical programming for the single destination and the multi-destination case. We showed that these problems are reducible respectively to the single item and the multi-item Lot Sizing problems, which are known to be NP-Hard in presence of capacity constraints. The analogy with these very well known optimization problems, let us consider the possibility to solve the SOP by using some of the exact and approximated methods proposed in literature for the Lot Size models. Further direction of research can rely on the possibility of extensions of the model to schedule lines connecting more transit terminals.

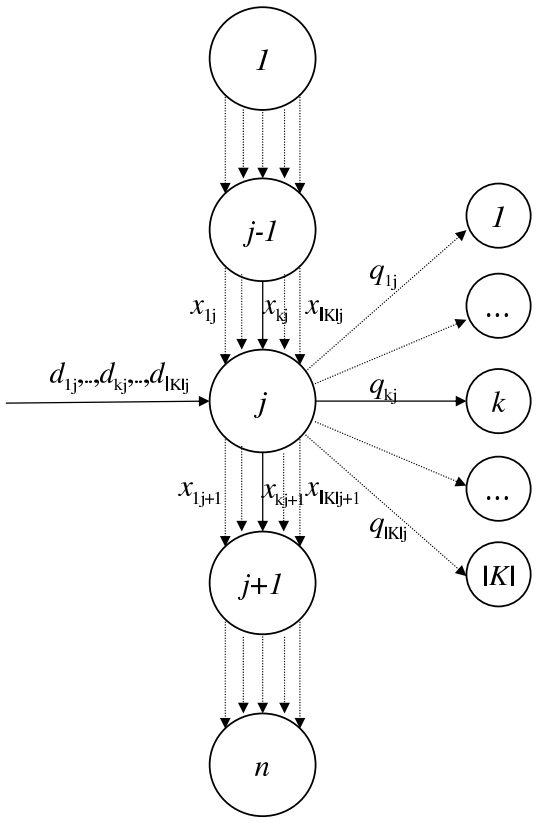


Figure 3: A scheme of the set of variables in the multi-destination model.

References

[1] A. Adamski (1995), Transfer optimization in public transport, in: J.R. Daduna et al. (eds.), Computer – Aided transit scheduling. Proceedings of the 6th international workshop on computer – aided scheduling of public transport, Lisbon, Portugal, July 6 – 9, 1993. Berlin: Springer Verlag, *Lect. Notes Econ. Math. Syst.* 430, 23 – 38.

[2] G.R. Bitram, H.H. Yanasse (1982), Computational complexity of the capacitated lot size problem, *Management Science* **28**, 1174 – 1186.

[3] A. Ceder, B. Golany, O. Tal (2001), Creating bus timetables with maximal synchronization, *Transportation Research Part A* **35**, 913 - 928.

[4] A. Ceder, O. Tal (1999), Timetable Synchronization for Buses, in: Wilson NHM (ed), Computer – Aided Transit Scheduling, *Lecture Notes in Economics and Mathematical Systems* **471**, Springer, Berlin, 245 – 258.

[5] J.R. Daduna, J.M.P. Paixao (1995), Vehicle Scheduling for Public Mass Transit, in: J.R. Daduna, I. Branco, J.M.P. Paixao (eds), Computer – Aided Scheduling of Public Transport, *Lecture Notes in Economics and Mathematical Systems* **430**, Springer, Berlin, 76 – 90.

[6] J.R. Daduna, S. Voss (1995), Practical Experiences in Schedule Synchronization, in: J.R. Daduna, I. Branco, J.M.P. Paixao (eds), Computer – Aided Scheduling of Public Transport, *Lecture Notes in Economics and Mathematical Systems* **430**, Springer, Berlin, 39 – 55.

[7] A. Desilets, J.M. Rousseau (1992), SYNCRO: A Computer – Assisted Tool for the Synchronization of Transfers in Public Transit Networks, in: M. Desrochers, J.M. Rousseau(eds), *Computer – Aided Scheduling of Public Transport*, Springer - Verlag, 153 – 166.

[8] A. Drexl, A. Kimms (1997), Lot sizing and scheduling — survey and extensions, *European Journal of Operational Research* **99**(2), 221 – 235.

[9] W. Domschke (1989), Schedule synchronization for public transit networks, *OR Sprektrum* **11**, 17 – 24.

[10] M. Florian, J.K. Lenstra, A.H.G. Rinnooy Kan (1980), Deterministic production planning algorithms and complexity, *Management Science* **26**(7), 669 – 679.

[11] C. Fleurent, R. Lessard, L. Seguin (2004), Transit Timetable Synchronization: Evaluation and Optimization, *Proceedings of the 9th International Conference on Computer – Aided Scheduling of Public Transport*, San Diego CA (U.S.A.), August 9-11.

[12] B. Karimi, S.M.T. Fatemi Ghomi, J.M. Wilson (2003), The capacitated lot sizing problem: a review of models and algorithms, *Omega* **31**(55) (Oxford), Elsevier, 365 – 378.

[13] W. Keudel (1988), Computer – aided line network design (DIANA) and minimization of transfer times in networks (FABIAN), in: Daduna, J.R., Wren, A., (eds), Computer – Aided Transit Scheduling, *Lecture Notes in Economics and Mathematical Systems* **308**, 315 – 326, Springer, Berlin.

[14] W.D. Klemt, W. Stemme (1988), Schedule synchronization for public transit networks, in: Daduna, J.R., Wren, A., (eds), Computer – Aided Transit Scheduling, *Lecture Notes in Economics and Mathematical Systems* **308**, 327 – 335, Springer, Berlin.

[15] J. Krarup, P.M. Pruzan (1983), Simple plant location problem: Survey and synthesis, *European Journal of Operation Research* **12**(11), 36 – 81.

[16] C.S. Revelle, G. Laporte (1996), The Plant Location Problem: New Models and Research Prospects, *Operations Research* **44**(6), 864 – 874.

[17] M. Schroder, I. Solchenbach (2006), Optimization of transfer quality in regional public transit, Technical Report, Fraunhofer ITWM, 29 S.

[18] H.M. Wagner, T.M. Within (1958), Dynamic version of the economic lot size model, *Management Science* **5**, 89 – 96.

[19] L.A. Wolsey (1995), Progress with single – item lot – sizing, *European Journal of Operational Research* **86**, 395 – 401.

[20] R.C.W. Wong, J.M.Y. Leung (2004), Timetable synchronization for mass transit railway, *9th International Conference on Computer - Aided Scheduling of Public Transport*, San Diego, California, August 9 - 11.

[21] S. Voss (1992), Network design formulation in schedule synchronization, in: Desrochers M., Rousseau J.M. (eds), *Computer - Aided Scheduling of Public Transport*, Springer – Verlag, 137 – 152.

A Decomposition Heuristic for Planning and Scheduling of Jobs on Unrelated Parallel Machines

Christoph Habla, Lars Mönch
University of Hagen, Department of Mathematics and Computer Science, 58097 Hagen, Germany
{Christoph.Habla, Lars.Moench}@FernUni-Hagen.de
Michele E. Pfund, John W. Fowler
Arizona State University, Department of Industrial Engineering, Tempe, AZ 85287-5906, USA
{Michele.Pfund, John.Fowler}@asu.edu

In this paper, we study a planning and scheduling problem for unrelated parallel machines. There are *n* jobs that have to be assigned and sequenced on *m* unrelated parallel machines. Each job has a weight that represents the priority of the corresponding customer order, a given due date, and a release date. An Automated Guided Vehicle is used to transport at maximum $Load_{max}$ jobs into a storage space in front of the machines in a given period of time. We consider *T* consecutive periods of time, and are interested in minimizing the total weighted tardiness (TWT) of the jobs across the *T* periods. To solve the problem, we present a mixed integer program (MIP) and a heuristic decomposition methodology. These methodologies are tested using stochastically generated test instances and compared. Results indicate that the decomposition approach performs comparably to the MIP while having reasonable solution times.

Keywords: Planning and Scheduling, Parallel Machines, Decomposition, Genetic Algorithms.

1. Introduction

Scheduling jobs in manufacturing is still challenging due to the NP-hardness of many scheduling problems, a lack of data, machine breakdowns, and multiple criteria [5,10]. Therefore, the development of efficient planning and control strategies is highly desirable for complex manufacturing systems. In the course of the development of new planning and control algorithms, the researchers and developers have to take into account the new opportunities of advanced software and hardware technologies.

In this paper, we model a planning and scheduling problem for unrelated parallel machines in a Printed Wiring Board (PWB) manufacturing environment. The process of PWB consists of multiple stages of production. Each board must pass through a pre-assigned sequence of manufacturing steps [16]. Scheduling jobs on parallel machines is a building block for any enterprise-wide scheduling approach for a PWB line. In the motivating PWB line, Automated Guided Vehicles (AGV's) are used to transport lots from one stage to another one.

We propose a multi-period formulation and show that the model can be solved very efficiently by means of genetic algorithms and dispatching rules. The multi-period formulation is a generalization of previous work done by Pfund *et al*. [11,16].

The paper is organized as follows. We describe the problem, related literature, and a MIP formulation in Section 2. Section 3 presents a decomposition approach that uses dispatching rules and genetic algorithms within the different phases of the decomposition. We show results of computational experiments in Section 4.

2. Problem and model

2.1. Problem Description

We consider a planning and scheduling problem for jobs on unrelated parallel machines in a PWB manufacturing environment. We assume that we know in advance which of the n jobs can be processed on the machines within a given horizon *T*. An AGV is used to transport at maximum $Load_{max}$

jobs each *P* hours to the machines. For simplicity reasons we assume that *T* is divided into t_{max} periods of size *P*. Therefore, the relation $T = Pt_{max}$ holds. We are interested in minimizing the total weighted tardiness (TWT) of the *n* jobs. The quantity TWT is defined as follows:

$$TWT = \sum_{j=1}^{n} w_j T_j = \sum_{j=1}^{n} w_j \left(c_j - d_j\right)^+. \tag{1}$$

Here we use for abbreviation the notation $x^+ := max(x,0)$. The customer specific priority of job *j* is denoted by w_j. The notation c_j is used for the completion time of job *j* whereas d_j denotes the due date of job *j*. The problem can be represented as

$$R_m \mid r_j, Load_{max} \mid TWT \tag{2}$$

applying the $(\alpha \mid \beta \mid \gamma)$ classification scheme for scheduling problems (cf. [4,3]). We use the notation R_m for unrelated parallel machines, i.e., the speed of the machines can be different and depends on the jobs (cf. [13] for the different types of parallel machine environments). We consider a dynamic planning and scheduling problem, i.e., each job has a ready time r_j. Throughout the rest of the paper, the ready time of a job is given by the period where the job can be processed for the first time.
The considered problem is NP-hard because the single machine problem $1||TWT$ [6] is also NP-hard. Therefore, efficient heuristics are needed to solve reasonable sized problems.
We finally summarize the decisions that have to be made within a period *t* where $1 \le t \le t_{max}$:

- **Load decisions** for the guided vehicles: selection of maximum $Load_{max}$ jobs from the set of non-planned jobs,
- **Assignment decisions:** assignment of jobs to a specific machine,
- **Sequencing decisions:** after the assignment of jobs to machines the sequence of the jobs on each machine has to be determined. This can be done by determining the position of the jobs on a single machine.

We call the problem to be researched a planning and scheduling problem because we consider future time periods. This leads to a planning problem. When we take into account only a single period, we have to deal with a pure scheduling problem.

2.2. Related Literature

Unrelated parallel machine scheduling problems are widely discussed in the scheduling literature. We refer, for example, to the recent survey paper by Pfund, Fowler, and Gupta [12].
An efficient branch & bound algorithm that is based on several dominance rules is described in [7]. However, only a single-period problem is presented. A related problem for unrelated parallel machines in a PWB environment is solved in [16,11] wherein a MIP formulation is given and the MIP is solved by Lagrange relaxation. However, only a single period is considered.
Decomposition techniques are widely used to solve complex scheduling problems (cf. [10] for different decomposition techniques). Different multi-phase decomposition approaches for scheduling jobs on parallel batch machines are discussed in [9]. The authors consider the phase batch formation, assignment of batches to single machines, and finally a sequencing phase for each single machine. Genetic algorithms are used for the assignment phase and dispatching rules are used for the remaining phases. We solve the planning and scheduling problem in this paper by a similar decomposition approach.

2.3. Mixed Integer Programming Formulation

In this section, we present a mixed integer programming formulation of our planning and scheduling problem. We introduce first the necessary index sets. The index sets can be derived from the load, assignment, and sequencing decisions. We consider therefore the following index sets:

$j \in J$:	jobs,	$1,\ldots, j_{max}$,
$i \in I$:	machines,	$1,\ldots,i_{max}$,
$t \in T$:	periods,	$1,\ldots,t_{max}$,
$k \in K^{(i)}$:	positions,	$1,\ldots,k_{max}^{(i)}$,
$k_{max}^{(i)}$:	maximum number of jobs per period on machine *i*.	

We consider the set $T^{(j)} := \left\{t \mid r_j \le t \le t_{max}\right\}$ for each $j \in J$. Furthermore, we introduce the index set $J^{(t)} := \left\{j \mid j \in J \wedge r_j \le t\right\}$ for each $t \in T$.
The model contains the following parameters:

P:	duration of a period,
$p_{j,i}$:	processing time of job *j* on machine *i*,
a_i:	time of the first availability of machine *i* in period *t=1*.

The already defined quantities d_j, w_j, and $Load_{max}$ are further model parameters. We use the following decision variables within the model. We define the binary variable

$$x_{j,i,t,k} := \begin{cases} 1, when\ job\ j\ is\ on\ machine\ i\ in\ period\ t\ at\ position\ k \\ 0, otherwise \end{cases} \tag{3}$$

for all $j \in J$, $i \in I$, $t \in T^{(j)}$, and $k \in K^{(i)}$. The variable $o_{i,t}$ measures for all $i \in I$ and $t \in T$ the overload on machine *i* in period *t*, i.e., the amount of time that is necessary to complete the last job on machine *i* in period *t* after the end of period *t*. The decision variable $c_{i,t,k}$ models the completion time of the job at position *k* in period *t* on machine *i* for all $i \in I$, $t \in T$, $k \in K^{(i)}$. The tardiness of job *j* is given by $T_j := \left(c_j - d_j\right)^+$ for all jobs $j \in J$. Note that the quantity $x_{j,i,k,t}$ is a binary variable whereas the remaining variables are real and non negative.
The objective function TWT has to be minimized. The optimization problem can be formulated as follows:

$$\text{(P)} \qquad \min \sum_{j \in J} w_j T_j \tag{4}$$

subject to the following constraints

$$\sum_{i \in I} \sum_{t \in T^{(j)}} \sum_{k \in K^{(i)}} x_{j,i,t,k} \quad = \quad 1, \quad \forall j \in J, \tag{5}$$

$$\sum_{j \in J^{(t)}} x_{j,i,t,k} \quad \le \quad 1, \qquad \forall i \in I, t \in T, k \in K^{(i)}, \tag{6}$$

$$c_{i,t,k} = a_i + \sum_{n=1}^{k} \sum_{j \in J^{(t)}} x_{j,i,t=1,n}\, p_{j,i}, \quad \forall i \in I, t = 1, k \in K^{(i)}, \tag{7}$$

$$c_{i,t,k} = (t-1)P + o_{i,t-1} + \sum_{n=1}^{k} \sum_{j \in J^{(t)}} x_{j,i,t,n}\, p_{j,i}, \quad \forall i \in I, t = 2,\ldots,t_{max}, k \in K^{(i)}, \tag{8}$$

$$c_{i,t,k} - p_{j,i} \leq tP + t_{\max} P(1 - x_{j,i,t,k}), \quad \forall j \in J, i \in I, t \in T^{(j)}, k \in K^{(i)}, \tag{9}$$

$$o_{i,t} \geq c_{i,t,k} - tP, \quad \forall i \in I, t \in T, k = k_{max}^{(i)} \tag{10}$$

$$T_j \geq c_{i,t,k} - d_j - (t_{max} + 1)P(1 - x_{j,i,t,k}), \quad \forall j \in J, i \in I, t \in T^{(j)}, k \in K^{(i)}, \tag{11}$$

$$\sum_{j \in J^{(t)}} \sum_{i \in I} \sum_{k \in K^{(i)}} x_{j,i,t,k} \leq Load_{max}, \quad \forall t \in T. \tag{12}$$

Constraint (5) assigns to each job a machine *i*, a period *t*, and a position *k*. Constraint (6) ensures that each position for each machine and period can be occupied only by at most one job. Constraints (7) and (8) calculate the completion time of the job at position *k* on machine *i* in period *t*. Constraint (9) ensures that the last job in period *t* on machine *i* starts with processing before period *t* is over. Constraint (10) determines the overtime in period *t* on machine *i*. Inequality (11) allows calculating the tardiness of job *j*. Constraint (12) makes sure that the maximum transportation quantity (capacity) of jobs for the AGV for each period is not violated.

3. Decomposition approach

Since the problem is NP-hard we develop a heuristic approach for the solution of large problem sizes. We suggest a three-phase decomposition algorithm:

- assignment of jobs to periods (load phase),
- assignment of jobs to the parallel machines for each single period in the second phase (assignment phase),
- solution of the resulting single machine sequencing problems of the second phase in the third phase (sequencing phase).

3.1. Load phase

The aim of the load phase consists of assigning $Load_{max}$ jobs to single periods. Because we are interested in minimizing TWT we use the Apparent Tardiness Cost (ATC) dispatching rule [14] to select appropriate jobs for each period. We calculate for each job *j* the ATC index given by

$$I_j(t) := \frac{w_j}{p_{j,i}} exp\left(- \frac{\left(d_j - p_{j,i} - t\right)^+}{k\bar{p}} \right), \tag{13}$$

where we denote by k a look-ahead parameter that is varied between 0.1 and 6.5. We use the notation $\bar{p}$ for average processing time of the remaining jobs. The time *t* is the time when any of the available machines becomes free in a simple deterministic forward simulation. At this time *t*, all available jobs that are ready in the period are ordered in descending order of their ATC index and the job with the highest ATC index is assigned to the available machine. Note that we have to perform the forward simulation to obtain a dynamic version of the ATC rule by determining appropri-

ate values for *t*. We consider the next period after selecting $Load_{max}$ jobs. Note that it is crucial for the performance of the ATC rule to select appropriate *k* values [13]. In order to find appropriate values we run the pure dispatching based approach described in Section 3.4 for a large number of *k* from an equidistant grid on $[0.1,6.5]$.

3.2. Assignment phase

We use a genetic algorithm (GA) in order to assign the jobs in each single period to the parallel machines, evaluate each of the assignments by aggregating the total weighted tardiness of sequences (obtained on each machine by using sequencing heuristics for the single machine case) on all machines. The GA converges towards assignments that give good solutions. GA's are widely used for hard combinatorial optimization problems (cf. [2] for applications in manufacturing).
We used a job-based chromosome representation in the GA. A solution of the problem to assign jobs to machines is an array whose length is equal to the number of jobs. The *s*-th element of the array represents the machine that is used to process job *s*. Each chromosome is randomly initialized by generating a random number from $\{1,...,m\}$ for each single gene. A standard one-point crossover operator [8] is used for performing the crossover operations. According to a user-specific probability, a gene of a chromosome is randomly changed to a different machine number for mutation purposes. Each chromosome is evaluated with the sum of the total weighted tardiness value of the jobs in a given period and a second term that represents the overload on the machines in the period. After the jobs are assigned to machines, each machine is sequenced and the sum of the total weighted tardiness values of all machines and the overload for each machine and period is used as the fitness function of the GA. A steady state GA with overlapping populations is used (cf. [8]). The GA is controlled by a prescribed number of generations. The parameter setting of the GA is done by computational experiments.

3.3. Sequencing phase

For sequencing the jobs on each single machine within each single period we use again the ATC rule. We apply again deterministic forward simulation as described for the load phase to update the time *t* in the ATC index in a correct way.

3.4. Pure dispatching rule based approach

A simple dispatching rule based scheme can be constructed using the ATC rule as described in Section 3.1. After the assignment of at maximum $Load_{max}$ jobs to each period we use a list based scheduling approach to assign jobs to parallel machines. Note that we simultaneously determine a sequence for each single machine. As already described we have to iterate over a large number of *k* values to find values that lead to small TWT values. We denote the pure dispatching rule based approach by ATCH. The suggested decomposition approach is denoted by GA-ATC.

4. Computational results

4.1. Design of experiments

We use the experimental design shown in Table 1 for our computational experiments. The applied test data were generated according to an extension of the test data description given in [1] to the case of parallel machines and multiple periods. The test instances depend on the number of parallel machines, the number of jobs in each period on a machine, and the ready time and due setting mechanism. Totally, we solve 135 test instances.
All experiments were performed on a Pentium IV, 3.4 GHz PC. We used the object-oriented framework GaLib (cf. [15]) in order to implement the suggested GA in an efficient way. The computing time was less than 20 seconds for GA-ATC and less than 0.3 seconds for ATCH for all test instances analyzed in Section 4.2. That is both fast with respect to the planning horizon of several hours. We take six independent replications of the GA runs for one test instance to obtain stochastically significant results.

Table 1. Factorial Design

Factor	Level	Count
Capacity of the AGV $Load_{max}$	{18,24,30} for {3,4,5} machines	1
Number of Machines $i_{\max}$	{3,4,5}	3
Number of Periods t_{max}	6	1
Average Number of Jobs per Period and Machine	4 $j_{max} = 72,96,120$	1
Length of a Period *P*	240	1
Efficiency v_i of the Machines	0.5 $(i = 1)$ 1.0 $(1 < i < i_{max})$ 1.25 $(i = i_{max})$	1
Processing Time of the Jobs for the Different Products (each of them with equal probability)	30, 45, 60 und 75	1
Ready Time	$r_j \sim 1 + \lfloor t_{max} * \alpha * U(0,1) \rfloor$ $\alpha = 0.25,0.50,0.75$	3
Due Date	$d_j \sim \left(r_j + \lceil t_{max} * \beta * U(0,1) \rceil\right) * P$ $\beta = 0.25,0.50,0.75$	3
Weight of the Jobs	$w_j \sim U(0,1)$	1
Number of Parameter Combinations		27
Number of Independent Problem Instances		5
Total Number of Problem Instances		135

4.2. Results of computational experiments

The results of the experiments are given in Table 2. We show the improvement ratio for the TWT value of the suggested decomposition approach to the TWT value of ATCH. Clearly, GA-ATC outperforms ATCH. As can be seen from the results the algorithm is sensitive to the ready time range that is controlled by α and the due date range that depends on the choice of β. Instead of comparing all test instances individually, the test instances where grouped according to the level of factors. For example, the results for *m=3* are for all runs with three machines while all other factors have been varied at their different levels.

Table 2. Computational Results for GA-ATC

	ATCH	**GA-ATC**
m = 3	1.00	0.96
m = 4	1.00	0.94
m = 5	1.00	0.92
α = 0.25	1.00	0.96
α = 0.50	1.00	0.96
α = 0.75	1.00	0.90
β = 0.25	1.00	0.97
β = 0.50	1.00	0.89
β = 0.75	1.00	0.96
Overall	1.00	0.94

In Table 3, we study the effect of different release and due date setting schemes. Therefore, we consider all possible α and β combinations. The choice of $\beta = 0.75$ leads to the largest improvement rates for $\alpha = 0.25$ because a larger β value causes a wider spread of due dates. In this situation the GA offers a lot of advantage. Larger values of α in combination with $\beta = 0.50$ also lead to large improvements.

Table 3. Effect of Release Date and Due Date Factor Setting

	ATCH	**GA-ATC**
$\alpha = 0.25,\ \beta = 0.25$	1.00	0.98
$\alpha = 0.25,\ \beta = 0.50$	1.00	0.97
$\alpha = 0.25,\ \beta = 0.75$	1.00	0.92
$\alpha = 0.50,\ \beta = 0.25$	1.00	0.98
$\alpha = 0.50,\ \beta = 0.50$	1.00	0.94
$\alpha = 0.50,\ \beta = 0.75$	1.00	0.97
$\alpha = 0.75,\ \beta = 0.25$	1.00	0.94
$\alpha = 0.75,\ \beta = 0.50$	1.00	0.75
$\alpha = 0.75,\ \beta = 0.75$	1.00	1.00

In Table 4, we compare the solution quality of ATCH and GA-ATC with the solution quality obtained by an ILOG CPLEX implementation of the MIP presented in this paper. The last column shows the improvement rates obtained by the ILOG software compared to ATCH and GA-ATC. We use only small size test instances (24 jobs) for $m \in \{2,3\}$. The ILOG solver runs at maximum for two hours.

Table 4. TWT Values for ATCH, GA-ATC, and ILOG

α	β	***M***	**ATCH**	**GA-ATC**	**ILOG -MIP (2 h)**	**Improvement [%]**	
						ATCH	GA-ATC
0.25	0.25	2	606	579	579	4	0
0.25	0.25	3	933	895	898	4	0
0.25	0.50	2	314	280	280	11	0
0.25	0.50	3	179	151	148	17	2
0.50	0.25	2	146	140	139	4	0
0.50	0.25	3	219	204	204	7	0
0.50	0.50	2	84	78	78	7	0
0.50	0.50	3	58	51	52	10	0

We see from Table 4 that GA-ATC is able to find solutions with the same TWT value as obtained by the ILOG solver in almost all cases. But even ATCH performs quite well. Note that it is not possible to solve test instances from Table 1 with a reasonable amount of time by using the ILOG software.

5. Conclusion

In this paper, we presented a problem description for a combined planning and scheduling problem. We developed a mixed-integer programming formulation. Because of the computationally intractable mixed integer programs, we suggested a decomposition heuristic based on hybridized genetic algorithms. Based on stochastically generated test instances it turned out that the decompo-

sition approach outperforms a simple dispatching rule based approach. We found that even the dispatching rule based approach provided near to optimal solutions for small size test instances based on comparison with solutions obtained by the ILOG software with a two hour maximum computation time. There are several directions for future research. First of all it seems to be possible to study different modifications of the genetic algorithm. It seems to be possible to avoid the load phase in the beginning. In this case, first an assignment step for jobs to machines is necessary. Then, in a second step it is required to assign the jobs to periods and sequence them on each single machine. However, a careful investigation is necessary because of the considerably larger search space of the GA in this case. Instead of looking for TWT it seems to be possible to consider other performance measures and to look for different product mix scenarios. Due to the parallel structure of the problem, the second future research direction is given by using column generation techniques where the columns represent a job assignment to one of the parallel machines. Here, appropriate sub problem solution techniques have to be developed. However, carrying out all the details is part of future research.

References

[1] M. S. Akturk, D. Ozdemir (2001), A New Dominance Rule to Minimize Total Weighted Tardiness with Unequal Release Dates, *European Journal of Operational Research* **135**, 394–412.

[2] H. Aytug, M. Khouja, F. E. Vergara (2003), Use of Genetic Algorithms to Solve Production and Operations Management Problems: a Review, *International Journal of Production Research* **41**(17), 3955–4009.

[3] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, J. Weglarz (2001), *Scheduling Computer and Manufacturing Processes*, 2nd edition, Springer, Berlin.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan (1997), Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, *Annals of Discrete Mathematics* **5**, 287–326.

[5] S. Kreipl, M. Pinedo (2004), Planning and Scheduling in Supply Chains: An Overview of Issues in Practice, *Production and Operations Management* **13**(1), 77–92.

[6] E. L. Lawler (1977), A "Pseudopolynomial" Time Algorithm for Sequencing Jobs to Minimize TotalWeighted Tardiness, *Annals of Discrete Mathematics* **1**, 331–342.

[7] C.-F. Liaw, Y.-K. Lin, C.-F. Cheng, M. Chen (2003), Scheduling Unrelated Parallel Machines to Minimize Total Weighted Tardiness, *Computers & Operations Research* **30**, 1777–1789.

[8] Z. Michalewicz (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edition, Springer, Berlin, Heidelberg, New York.

[9] L. Mönch, H. Balasubramanian, J. W. Fowler, M. E. Pfund (2005), Heuristic Scheduling of Jobs on Parallel Batch Machines with Incompatible Job Families and Unequal Ready Times, *Computers & Operations Research* **32**, 2731–2750.

[10] I. M. Ovacik, R. Uzsoy (1997), *Decomposition Methods for Complex Factory Scheduling Problems*, Kluwer Academic Publishers, Boston.

[11] M. E. Pfund (2002), *Evaluation of Uncertainty on Scheduling Algorithms in Printed Wiring Board Manufacturing*, PhD Dissertation, Arizona State University, Department of Industrial Engineering.

[12] M. E. Pfund, J. W. Fowler, J. N. D. Gupta (2004), A Survey of Algorithms for Single and Multi-objective Unrelated Parallel-Machine Deterministic Scheduling Problems, *Journal of the Chinese Institute of Industrial Engineers* **21**(3), 230–241.

[13] M. Pinedo (2002), *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Second Edition, Englewood Cliffs, NJ.

[14] A. Vepsalainen, T. E. Morton (1987), Priority Rules and Lead Time Estimates for Job Shop Scheduling with Weighted Tardiness Costs, *Management Science* **33**, 1036–1047.

[15] M. Wall (2007) *GaLib – a C++ Library of Genetic Algorithm Components*, http://lancet.mit.edu/ga/.

[16] L. Yu, H. M. Shih, M. E. Pfund, W. M. Carlyle, J. W. Fowler (2002), Scheduling of Unrelated Parallel Machines: an Application to PWB Manufacturing, *IIE Transactions* **32**, 921–931.

Scheduling Steel Production using Mixed-Integer Programming and Constraint Programming

Andrew Davenport, Jayant Kalagnanam
IBM T.J.Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, NY 10598, USA,
{davenport, jayant}@us.ibm.com

We present an overview of a system we have developed for generating short-term production schedules for a large steel manufacturer. We have developed a decomposition based approach which uses mixed-integer programming to generate a plan for production on the downstream casting processes, and constraint programming to generate a schedule upstream of these casting processes, taking into account detailed scheduling constraints and preferences. One drawback to this approach is that downstream production plans can be generated which cannot be scheduled on the upstream processes, since downstream planning does not take into account bottlenecks that may occur when scheduling upstream. We investigate an integration mechanism, which uses constraint programming to build local estimations of upstream capacity utilization, which are used in the integer programming formulation for downstream production planning. We present results illustrating the effectiveness of this approach.

Keywords: Applications, Production Scheduling

1. Introduction

In this paper we present an overview of a system we have developed for generating short-term production schedules for a large steel manufacturer. The scheduling of steel production involves solving two related problems for the upstream and downstream processes of manufacturing. The downstream problem involves the selection and sequencing of groups of contiguous jobs on a number of machines subject to shift-level capacity constraints on the number of orders of each product type, tight inventory constraints and sequence-dependent setup times. The upstream processes are scheduled on unary capacity resources subject to alternate recipes and resources for each job and precedence constraints with tight minimum and maximum time lags. We have investigated a decomposition-based approach that uses mixed-integer programming and constraint programming to schedule the downstream and upstream processes respectively. One drawback to this approach is that bottlenecks on the upstream processes are not taken into account during the downstream process scheduling. We discuss some techniques we have explored to take into account such bottlenecks in a decomposition-based approach to steel production scheduling.

In the sections that follow, we present an overview of the processes in steel manufacturing and discuss the formulation of the scheduling model. We then present the two-stage decomposition based approach and discuss integration strategies to improve its performance in the presence of upstream bottlenecks. Finally, we present some experimental results.

2. Steelmaking processes

The scope of the scheduling problem addressed by the system covers the production of solid steel slabs from molten iron. The main processes involved in the manufacturing of steel slabs are illustrated in Figure 1. The four main process areas are:

1. The *Blast Furnace* (far left in Figure 1) where iron is heated to a very high temperature to become molten. There is a continuous flow of molten iron from the blast furnace to the downstream processes. This flow is given as input to the scheduling problem formulation.
2. The *Basic Oxygen Furnace* is the first process that all production must pass through after leaving the blast furnace. At this process, molten iron starts to become differentiated with respect to grade and chemical composition.

3. The *Refining Processes* consist of a number of steps such as reheating, ladle furnace and stirring. Not all production will pass through all of these steps, and these steps are not ordered (the steps that are used depend on the chemical composition of the final products.)
4. The *Continuous Casters* (far right in Figure 1): All production passes from the refining stage to the final continuous casting process. In this process, molten steel is poured into a long, adjustable copper mold. As the steel passes through the mold, it is cooled by water jets and solidifies into slabs of a specific dimension.

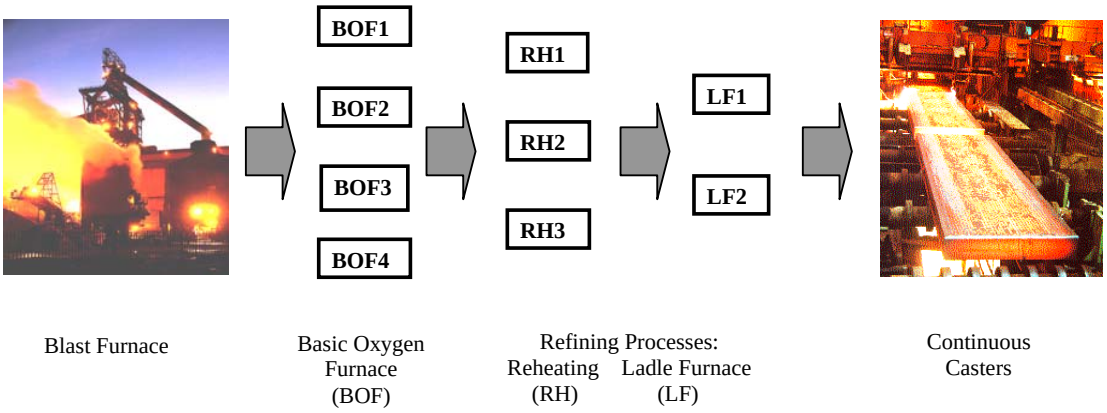


Figure 1: An illustration of the basic processes involved in the manufacturing of steel.

3. Scheduling model formulation

3.1 Components of the model

Steel is usually produced on a make to order basis. During the manufacturing of steel products, orders for steel slabs are batched into units of "charges". All distinct operations from the basic oxygen furnace and the refining process stages take place on a single charge of steel. At the continuous casting stage, operations take place on a sequence of (4-12) contiguous charges, which is called a "cast". The batching of orders into charges and the sequencing of charges into casts is provided as input to the scheduling system. These batching and sequencing steps are tackled as complex, multi-criteria optimization problems, the descriptions of which are outside of the scope of this paper.

3.2 Model of a single cast

Figure 2 presents a Gantt chart view illustrating how the concept of charges and casts are reflected in the formulation of the scheduling model. The Figure shows the schedule for the operations involved in the production of a single cast of steel. The interesting aspects of this formulation to note are that:

1. Each set of operations, for example A_1, A_2, A_3 and A_4 in Figure 2, represent the set of operations required to produce a single charge of steel. This corresponds to a single job in the scheduling model. All activities are non-preemptible.
2. In the final casting process a cast is produced consisting of a sequence of contiguous charges. This sequence is given as input to the problem. The processing of consecutive charges in a single cast on the casting processes must be without interruption. In Figure 2, the operations A_4, B_3 and C_2 are scheduled as a cast on the casting process. Hence the start time of operation B_3 must occur at the end time of operation A_4, and the start time of operation C_2 must occur at the end time of operation B_3.

3. There are tight minimal and maximal time lag constraints between consecutive operations in the same job[1]. For instance the maximum time lag between the end of A_1 and start of A_2 is 20 minutes.
4. At each process there are a number of machines that can be used to process an operation. The scheduler determines which machine an operation is assigned to[2]. Each machine has different operating characteristics and a different physical location. As a result, the processing time of an operation at a process will depend on which machine it is assigned to.
5. For each charge we are given a preferred recipe specifying a sequence of process steps that the charge must pass through. We are also given between 0 and 3 alternate recipes that can be used. In practice, most (90-95%) of charges will be assigned to their preferred recipes.

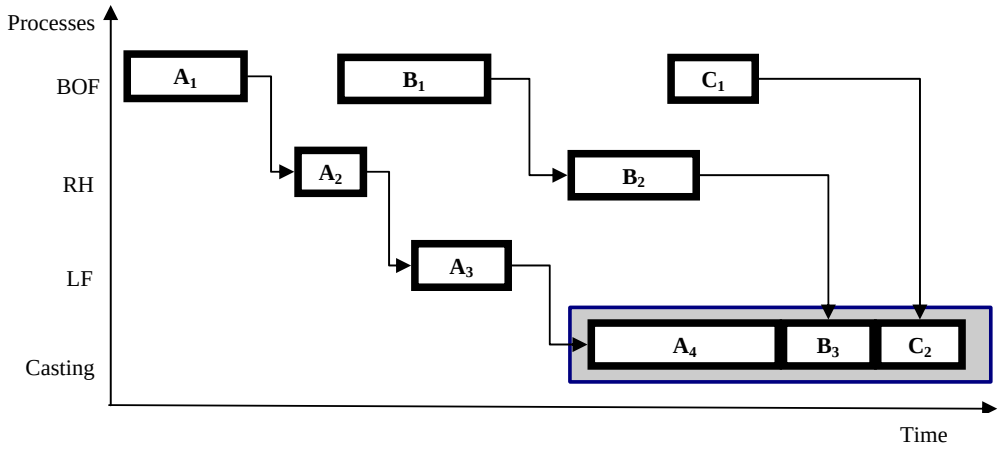


Figure 2: A Gantt chart illustration of the operations (activities) involved in manufacturing a single cast in a steel plant, from the basic oxygen furnace (BOF) to the refining processes (reheating (RH), and ladle furnace (LF)) to the casting process.

3.3 Hot metal inventory constraints

As we mentioned earlier, there is a continuous flow of molten iron from the blast furnace. This is specified as a problem input in terms of the number of tons per hour of molten iron flow. We consider some quantity of molten iron to be consumed in the first operation of each job at the basic oxygen furnace process. Between the blast furnace and the basic oxygen furnace there is finite capacity buffer, where the molten iron is stored until some operation is scheduled that consumes it. We have minimum and maximum inventory level constraints on the quantity of molten iron that can be allowed to accumulate in this buffer.

3.4 Model of cast scheduling

In section 3.2 we presented the basic scheduling model for the production of a single cast. In the full problem we are required to schedule many casts (60-100) on a number (3-9) of distinct casting machines over a 1-2 day horizon. The system is given as input a larger set of casts than can be scheduled within the horizon. The scheduler determines which casts to schedule in the short term subject to:

1. Shift level capacity constraints: each charge has attributes such as product type and grade. Constraints state the minimum and maximum number of charges that can be produced per shift by each attribute. (A shift is a period of 8 hours, and there are 3 shifts per day.)

[1] These arise as a result of the movement of the molten steel between processes. If the steel cools down, it is necessary to reheat it, which is expensive in terms of energy consumption.

[2] There is an exception for the casting process, where every charge in a cast executes on the same casting machine that is given as input to the scheduler.

2. Preferred start dates: we are required to maximize the number of charges that are processed on their preferred dates.
3. Sequence dependent setup times between consecutive casts processed on the same casting machine.
4. Minimum and maximum hot metal inventory level constraints.

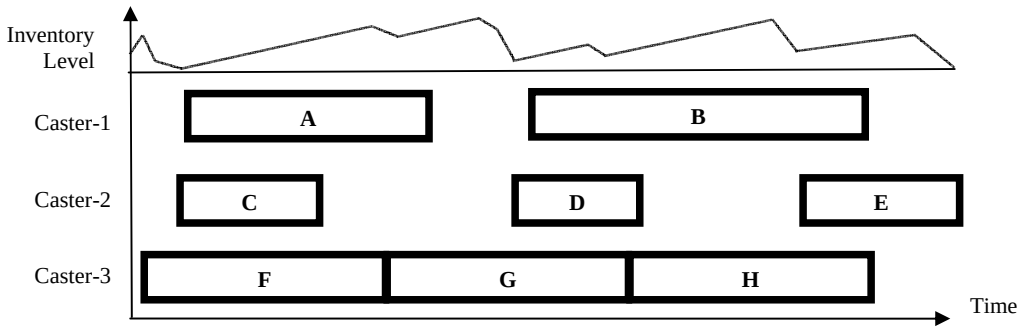


Figure 3: Illustration of a Gantt chart specifying a cast schedule.

The cast schedule specifies which casts are to be processed at what time on the available casting machines over the horizon of the schedule subject to constraints described in this section. Figure 3 illustrates a simple cast schedule for 3 casting machines. Note that on caster-3 it is possible to schedule three casts *F*, *G* and *H* consecutively with no setup time between them. Although there is no explicit objective to minimize setup time used in the schedule, in practice the maximum hot metal inventory constraint and constraints on the minimum number of charges to schedule per shift require us to use as little setup time as possible.

4. Solution Approach

4.1 Basic approach overview

The solution approach we have developed exploits the fact that we have fairly tight maximum time lag constraints between all consecutive pairs of operations in a single job for a charge (this may be as little as 20-40 minutes.) As a result, the schedule for a single cast over all processes will necessarily be localized in time. Our solution approach decomposes the full problem into two subproblems that are solved sequentially:

1. ***Downstream cast scheduling***: Cast scheduling determines which casts we are going to schedule and when; satisfying hot metal inventory constraints, shift level capacity constraints, sequence-dependent setup times between casts and preferred start times of casts. We do not consider the scheduling of any upstream processes of casting at this stage. We model this problem using a time-indexed integer programming formulation with a time granularity of 15-30 minutes and solve it using ILOG CPLEX[3].
2. ***Upstream detailed scheduling***: From the solution of the cast-scheduling problem we create a constraint-programming model for scheduling the processes upstream of casting, taking into account detailed scheduling constraints and preferences (such as minimum and maximum time lag between operations, resource exclusion constraints, state resource constraints, preferences on recipe and resource assignments). This model is formulated at a fine level of time granularity (30 seconds). Since the cast schedule has already been determined, we do not need to take into account any of the cast scheduling constraints in this model[4].

[3] For the purposes of cast scheduling, we assume all charges will be scheduled on their preferred route. The upstream detailed scheduling stage may reassign routes.

[4] The detailed scheduler is developed using a C++ constraint-programming library for manufacturing scheduling (currently called the "Watson Scheduling Library") developed at IBM Research.

The assumption we are making for this decomposition to work is that since the schedule for a single cast over all processes is localized in time (due to the maximum time lag constraints within a job), we can create the cast schedule for all casts independently of the processes upstream of the caster, and then use constraint programming to find a feasible upstream process schedule.

4.2 Basic cast scheduling integer programming model

We use a time-indexed formulation to generate a cast schedule, modeling the shift level capacity constraints and inventory constraints as side constraints. We divide the scheduling horizon into a set of equal, contiguous time periods (of usually between 15 and 30 minutes). Each time period is labeled by an index t taking a value from 1 to T. Let x_{it} be a decision variable whose value is 1 if cast i is scheduled to start processing in time period t, and whose value is 0 otherwise. Let p_i denote the number of time periods required to process cast i and let r_{ij} indicate the number of setup time periods between cast i and cast j if cast i directly follows cast j in the schedule. For each cast i we pick at most one starting time period:

$$\sum_{t=1}^{T-p_i+1} x_{it} \le 1 \quad \forall i \quad (x_{it} = 0 \ \forall t > T - p_i + 1) \tag{1}$$

For each caster C, given a set $S(C)$ of casts assigned to the caster, we can process at most one cast during each time period:

$$\sum_{i \in S(C)} \sum_{s=t-p_i+1}^{t} x_{is} \le 1 \quad \forall t \in \{1..T\} \tag{2}$$

The sequence-dependent setup time between each pair of casts i and j assigned to the same caster is modeled as follows:

$$\sum_{s=t+p_i}^{t+p_i+r_{ij}+1} x_{js} \le 1 - x_{it} \quad \forall i \forall j, i \ne j \ \ \forall t \in \{1..T - p_i - p_j + 1\} \tag{3}$$

We introduce indicator variables y_{ikts} associated with each cast i and time period t, having the value 1 if the k^{th} charge in cast i starts processing in time period s ($s \ge t$)[5]. The hot metal inventory constraint (stating that the inventory level is maintained within some limits) is modeled as follows[6]:

$$\begin{aligned} HM_{\min,t} \quad &\le \quad I_t \quad \le \quad HM_{\max,t} \quad \forall t \in \{1..T\} \\ I_t &= I_{t-1} + P - \sum_{s \in R(t)} y_{ikts} \times W_{ik} \quad \forall t \in \{1..T\} \end{aligned} \tag{4}$$

I_0 is the initial inventory level and P is the hourly production rate of inventory. $R(t)$ denotes the set of casts whose processing time periods intersects the time period t. W_{ik} is the amount of inventory consumed by the k^{th} charge of cast i. Finally, given a set of product types PT, where each product type $\alpha \in PT$ represents the set of jobs (charges) of product type α, and a set of shifts S, where each shift $\sigma \in S$ represents the set of time periods in shift σ, we model the capacity constraints on the minimum ($N_{\alpha\sigma,min}$) and maximum ($N_{\alpha\sigma,max}$) number of jobs of product type α in shift σ:

$$N_{\alpha\sigma,\min} \le \sum_{i,k \in \alpha} \sum_{t \in \sigma} y_{ikts} \le N_{\alpha\sigma,\max} \quad \forall \alpha \in PT, \forall \sigma \in S \tag{5}$$

The objective is usually to minimize the hot metal inventory level (subject to constraint 4).

4.3 Constraint programming model

Given a cast schedule, the goal of the detailed scheduler is to schedule all upstream processes taking into account detailed constraints and preferences[7]. A thorough description of the detailed

---

[5] Note if x_{it} is 0 then y_{ikts} is 0. This indicator variable is not strictly required (it can be directly deduced from each x_{it}) but is introduced here to help clarify the discussion.

[6] The constraint presented here models hot metal inventory being consumed at the casting process, in order to simplify the presentation. In the full model, we take into account the fact that the inventory is consumed further upstream by taking into account the lead-time between casting and the upstream processes.

scheduler is presented in [4]. To summarize, the scheduler is required to determine the recipes for each charge to follow in the schedule, the resources used by each job at each process stage, the sequencing of operations on each resource at each process stage subject to unary resource capacity constraints and temporal constraints, taking into account preferences on recipe and resource assignments. The scheduling solver is based on fairly well known techniques in constraint programming, such as those presented in [1,5]. The branching in the solver is based on the precedence constraint-posting framework described in [3] for sequencing operations, and simple texture-measurement based heuristics for recipe and resource assignment selection [2]. We use timetable and disjunctive resource constraint propagation [1]. Temporal constraint propagation is performed using a variation of the incremental longest-paths algorithm developed in [6].

4.4 Integration issues

One drawback of our decomposition-based approach arises from not taking into account upstream processes in the formulation of the downstream cast-scheduling problem; we encountered unforeseen bottlenecks on some of these upstream processes during detailed scheduling. Sometimes this results in the solution to the cast scheduling solution being infeasible on the upstream processes.

One solution is to extend the cast-scheduling model to perform some scheduling of the upstream processes. However, the time-indexed formulation of the cast-scheduling problem is at a relatively coarse level of time granularity (15-30 minutes), relative to the time granularity of the detailed scheduling constraints (at the 30 second level.) Using a finer time granularity in the cast-scheduling model in order to accommodate such constraints significantly increases the size of the model and the time taken to find a solution.

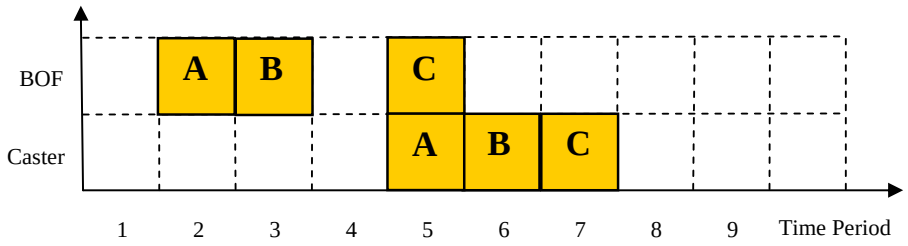


Figure 4: An illustration of the estimated capacity utilization on the upstream process BOF for a cast starting on the Caster process in time period 5.

We developed an alternative approach to avoiding upstream bottlenecks during cast scheduling by adding capacity constraints on the upstream processes to the cast-scheduling integer programming model. In order to formulate such capacity constraints, we need to be able to estimate for each cast how much capacity of the upstream processes they will utilize, and when. This is illustrated in Figure 4, where we represent part of the time-indexed cast-scheduling formulation involving a single cast of 3 charges, *A*, *B* and *C*, starting in time period 5 on the Caster and using 3 time periods (5-7) of Caster capacity. In this example if charge *A* in the cast starts in period 5 on the Caster, we estimate that it will use 1 time period of upstream BOF capacity in period 2. Note that upstream scheduling may later determine that the actual BOF capacity used by these charges is somewhere else in the schedule. However, since we have tight maximum time lag constraints between consecutive operations in a job for each charge, our working assumption is that we can estimate upstream capacity utilization that is accurate with respect to the final upstream schedule.

We generate a capacity utilization profile for each of the upstream bottleneck processes used by each of the charges in each cast[8]. From the capacity utilization profiles we can formulate a function *S(i,s,t,Pr)* which specifies the set of charges in cast *i* that (are estimated to) utilize some upstream process capacity *Pr* in time period *s*, given that the cast is scheduled to start processing in

[7] In practice we allow the detailed scheduler some flexibility to move casts on the casters subject to the cast sequence determined by the high level planning.
[8] Note that the processes upstream of casting do not have sequence dependent setup times.

time period t ($s \leq t$)[9]. We add capacity constraints to the cast-scheduling integer programming formulation for each time period t and upstream process Pr as follows:

$$y_{ikts} = x_{it} \qquad \forall i \; \forall s,t \in \{1..T\} \; \forall k \in S(i,s,t,PR) \; \forall \Pr$$
$$\sum_{i} \sum_{t=1}^{T-p_i+1} \sum_{k \in S(i,s,t,\Pr)} y_{ikts} \leq Cap(\Pr, s) \quad \forall s \in \{1..T\} \; \forall \Pr \qquad (6)$$

The term *Cap(Pr,s)* denotes the available capacity of the upstream process *Pr* in time period *s*.

We investigated two ways of estimating upstream capacity utilization. The first estimate (which we refer to as *min-lead-time*) is based on calculating the minimum lead-time for each charge in a cast between the upstream processes and casting process. For instance in Figure 4 this lead-time is 3 periods for charge *A* and is 1 period for charge *C*. We then construct a capacity profile based on the assumption that the lead-time is minimized between processes. An example of a capacity profile for real cast data using the *min-lead-time* estimate is illustrated in Figure 5.

The second estimate (which we refer to as *detailed-schedule*) calculates a measure of expected capacity utilization by generating a detailed schedule for each cast over all processes upstream of casting, independent of all other casts in the problem. This single cast schedule is found using the constraint programming based solver for upstream scheduling. An example of a capacity profile using the *min-lead-time* estimate for the same cast data as was used in Figure 5 is illustrated in Figure 6. Note that the *detailed-schedule* estimate here finds a capacity estimation that uses a slightly greater time extent than *min-lead-time*, but never uses more than 1 unit of capacity.

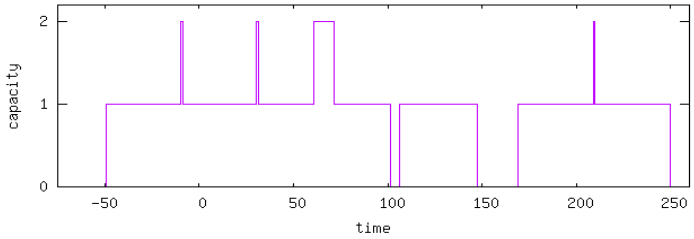


Figure 5: Min-lead-time upstream capacity estimation for a single cast.

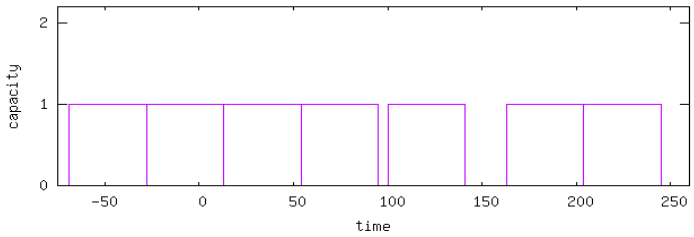


Figure 6: Detailed-schedule upstream capacity estimation for a single cast.

5. Experimental results

We present experimental results in Table 1 for 3 instances of real production data, comparing the *min-lead-time* and the *detailed-schedule* capacity utilization estimations. We compare the number of jobs scheduled by the cast scheduler over a 2-day horizon and the CPU time required by the cast scheduler to generate each plan. For these data sets, the upstream detailed scheduler was able to find a feasible schedule for all the casts in the cast schedule with a CPU time of usually less than 1 minute. Using the *detailed-schedule* capacity estimation, we were always able to find schedules with more jobs (charges) in the horizon than were found using the *min-lead-time* capacity estima-

[9] In practice it is usually sufficient to generate a single capacity utilization profile for a period of time, such as a shift, where the capacity constraints in the schedule do not change.

tion, using less CPU time. In practice, when using the *min-lead-time* capacity estimation we were usually not able to match the performance of the human experts in generating production schedules, with respect to number of jobs scheduled. With the *detailed-schedule* capacity estimation, we were able to significantly improve upon the performance of the human experts.

Instance	Number of Jobs (Charges) Scheduled		CPU Time (sec) for Cast Scheduling	
	min-lead-time	detailed-schedule	min-lead-time	detailed-schedule
1	122	138 (11.59%)	1406	160
2	162	163 (0.61%)	596	215
3	113	134 (15.67%)	158	146

Table 1: Experimental results for cast scheduling.

6. Conclusions

The scheduling of steel production involves solving two related problems for the upstream and downstream processes of manufacturing. The downstream, cast-scheduling problem requires the selection and sequencing of groups of contiguous jobs (casts) on a number of machines subject to shift-level capacity constraints, inter-process inventory constraints and sequence-dependent setup times. The upstream processes are scheduled on unary capacity resources subject to alternate recipes and resources for each job and precedence constraints with tight minimum and maximum wait times. We have presented a decomposition-based approach that uses mixed-integer programming to generate a production plan for downstream cast scheduling at a coarse level of time granularity, and constraint programming to schedule upstream processes subject to detailed scheduling constraints at a fine level of time granularity. In order to improve the performance of this approach, we found it necessary to take into account bottlenecks, which could appear on the upstream processes into the downstream planning model. We developed an approach for adding constraints to this downstream model by first generating a detailed upstream schedule for each cast in isolation using constraint programming, and using such schedules to estimate the capacity utilization for each cast on the upstream processes. We have presented a small set of experimental results that indicate this approach shows some promise.

Acknowledgements

The authors would like to thank Fatma Kilinc Karzan for comments on earlier drafts of this paper.

References

[1] P. Baptiste, C. LePape and W. Nuijten (2001), *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, International Series in Operations Research and Management Science, Vol. 39, Kluwer.

[2] J.C. Beck, A.J. Davenport, E.M. Sitarski and M.S. Fox (1997), Texture-Based Heuristics for Scheduling Revisited, *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, 241-248, AAAI Press / MIT Press.

[3] C.C. Cheng, and S.F. Smith (1996). Applying constraint satisfaction techniques to job shop scheduling, *Annals of Operations Research, Special Volume on Scheduling: Theory and Practice*, 1.

[4] A. Davenport, J. Kalagananam, C. Reddy, S. Siegel and J. Hou (2007), An application of constraint programming to generating detailed operations schedules for steel manufacturing, *Proc. 13th International Conference on Principles and Practice of Constraint Programming.*

[5] U. Dorndorf, E. Pesch, and P.-H. Toan, A Time-Oriented Branch-and-Bound Algorithm for Resource-Constrained Project Scheduling with Generalised Precedence Constraints", Management Science **46**(10), 1365 – 1384.

[6] I. Katriel and P. Van Hentenryck (2005), Maintaining Longest Paths in Cyclic Graphs, *Proc. 11th International Conference on Principles and Practice of Constraint Programming*.

Towards the Optimal Solution of the Multiprocessor Scheduling Problem with Communication Delays

Tatjana Davidović
Mathematical Institute, Serbian Academy of Sciences, Belgrade, Serbia, tanjad@mi.sanu.ac.yu
Leo Liberti
LIX Ecole Polytechnique, F-91128 Palaiseau, France, liberti@lix.polytechnique.fr
Nelson Maculan
COPPE – Systems Engineering, Federal University of Rio de Janeiro, Brazil, maculan@cos.ufrj.br
Nenad Mladenović
School of Information Systems, Computing and Mathematics, Brunel University, UB83PH Uxbridge UK, nenad.mladenovic@brunel.ac.uk

We consider the Multiprocessor Scheduling Problem with Communication Delays, where the delay is proportional to both the amount of exchanged data between pairs of dependent tasks and the distance between processors in the multiprocessor architecture. Although scheduling problems are usually solved by means of heuristics due to their large sizes, we propose methods to identify optimal solutions of small and medium-scale instances. A set of instances with known optima is a useful benchmarking tool for new heuristic algorithms. We propose two new Mixed-Integer Bilinear Programming formulations, we linearize them in two different ways, and test them with CPLEX 8.1. To decrease the time needed by CPLEX for finding the optimal solution, we use Variable Neighborhood Search heuristic to obtain a good approximation for the initial solution.

Keywords: Multiprocessors, Task Scheduling, Communication Delays, Linear Programming, CPLEX.

1 Introduction

In this paper we propose formulation-based and combinatorial methods to find optimal solutions to the Multiprocessor Scheduling Problem with Communication Delays (MSPCD). This consists in finding a minimum makespan schedule to run a set of tasks with fixed lengths L_i on an arbitrary network of homogeneous processors. The task precedence is modelled by a directed acyclic graph $G = (V, A)$ where V is the set of tasks and an arc (i, j) exists if task i has to be completed before task j can start. If i and j are executed on different processors $h, k \in P$, they incur a communication cost penalty γ_{ij}^{hk} dependent on the distance d_{hk} between the processors, and on the amount of data c_{ij} exchanged between the corresponding tasks ($\gamma_{ij}^{hk} = \Gamma c_{ij} d_{hk}$, where Γ is a known constant).

Since this problem is a super-class of the classic scheduling problem, it is NP-hard. Its natural application field is in compiler design [13] and robotics [14]. The size of real-life instances is almost always too large to be solved to optimality, so heuristic algorithms are used to find good solutions. When designing new heuristic methods, it is necessary to validate them on benchmark instances with known optimal solutions. The motivation of this study (and purpose of this paper) is to provide a tool to accomplish this task, i.e. finding optimal solutions of the MSPCD. Of course, our methods are prohibitively computationally expensive to be applied to real-life MSPCD instances, so our numerical experiments were carried out on a set of small- and medium-sized instances.

In the literature one can find mathematical formulations for different variants of scheduling problems. For example, in [12] the formulation for scheduling independent tasks is given. Scheduling of dependent tasks without communication is modelled in several different ways [1, 10]. In [9]

the authors considered the problem of scheduling dependent tasks onto heterogeneous multiprocessor architecture, but neglected communication time. To the best of our knowledge, up to now no mathematical programming model involving communications was developed and solved to optimality. Moreover, in existing models targeting similar problems, the processor topology is assumed to be a complete graph.

The main contributions of this paper are: (a) an adaptation of the classic scheduling formulation with ordering variables to the MSPCD, and a compact linearization of the resulting bilinear program; (b) a new formulation based on rectangle packing, which considerably reduces the number of variables; (c) comparative computational experiments on CPLEX 8.1 solving the two formulations with the impact of good, heuristic-based initial solution for the search process.

The rest of this paper is organized as follows. The two formulations are presented in Section 2. In Section 3 we discuss the comparative computational results. Section 4 concludes the paper.

2 Problem formulation

In this section we present two new formulations for the MSPCD. One is an adaptation of the "classic" scheduling model with variables expressing both assignment and ordering, and the other is derived from the idea of identifying tasks with rectangles and packing rectangles into a larger rectangle representing the total makespan and processor set. The parameters for both models are: the set of task lengths L_i ($i \in V$), the task precedence graph $G = (V, A)$, the costs c_{ij} on the arcs $(i, j) \in A$, the processor distance matrix (d_{kl}) for $k, l \in P$. We let $\delta^-(j)$ be the set of precedents of task i, that is $\delta^-(j) = \{i \in V \mid (i, j) \in A\}$.

2.1 Classic formulation

The classic scheduling formulation employs a set of binary variables which control both assignment of task to processor and position in the order of tasks executed by the given processor, and a set of continuous variables indicating the starting time for each task. Thus, we let:

$$\forall i, s \in V, k \in P \; y^s_{ik} = \begin{cases} 1 & \text{task } i \text{ is the } s\text{-th task on processor } k \\ 0 & \text{otherwise,} \end{cases}$$

and $t_i \in \mathbf{R}_+$ be the starting time of task i for all $i \in V$.

The MSPCD can be formulated as follows:

$$\min_{x,t} \quad \max_{i \in V} \{t_i + L_i\} \tag{1}$$

$$\forall\, i \in V \quad \sum_{k=1}^{p} \sum_{s=1}^{n} y^s_{ik} = 1 \tag{2}$$

$$\forall\, k \in P \quad \sum_{i=1}^{n} y^1_{ik} \leq 1 \tag{3}$$

$$\forall\, k \in P, s \in V \backslash \{1\} \quad \sum_{i=1}^{n} y^s_{ik} \leq \sum_{i=1}^{n} y^{s-1}_{ik} \tag{4}$$

$$\forall j \in V, i \in \delta^-(j) \quad t_j \geq t_i + L_i + \sum_{h=1}^{p} \sum_{s=1}^{n} \sum_{k=1}^{p} \sum_{r=1}^{n} \gamma^{hk}_{ij} y^s_{ih} y^r_{jk} \tag{5}$$

$$\forall k \in P, s \in V \backslash \{n\}, i, j \in V \quad t_j \geq t_i + L_i - M \left(2 - \left(y^s_{ik} + \sum_{r=s+1}^{n} y^r_{jk} \right) \right) \tag{6}$$

$$\forall i, s \in V, k \in P \qquad y_{ik}^{s} \in \{0,1\} \tag{7}$$
$$\forall i \in V \qquad t_i \geq 0, \tag{8}$$

where $M \gg 0$ is a sufficiently large penalty coefficient and γ_{ij}^{hk} represent the delay between tasks i and j as mentioned in Section 1.

Equations (2) ensure that each task is assigned to exactly one processor. Inequalities (3)-(4) state that each processor can not be simultaneously used by more than one task. (3) means that at most one task will be the first one at k, while (4) ensures that if some task is the s^{th} one ($s \geq 2$) scheduled to processor k then there must be another task assigned as $(s-1)$-th to the same processor. Inequalities (5) express precedence constraints together with communication time required for tasks assigned to different processors. Constraints (6) define the sequence of the starting times for the set of tasks assigned to the same processor. They express the fact that task j must start at least L_i time units after the beginning of task i whenever j is executed after i on the same processor k; the M parameter must be large enough so that constraint (6) is active only if i and j are executed on the same processor k and $r > s$.

2.1.1 Linearization of the classic formulation

The mathematical formulation of MSPCD given by (1)-(8) contains linear terms in the continuous t variables and linear and bilinear terms in the binary y variables. It therefore belongs to the class of mixed integer bilinear programs. It is possible to linearize this model by introducing a new set of (continuous) variables $\zeta_{ijhk}^{sr} \in [0,1]$ which replace the bilinear terms $y_{ih}^{s} y_{jk}^{r}$ in Eq. (5) [4, 5].

The linearizing variables ζ_{ijhk}^{sr} have to satisfy the following linearization constraints:

$$y_{ih}^{s} \geq \zeta_{ijhk}^{sr} \tag{9}$$
$$y_{jk}^{r} \geq \zeta_{ijhk}^{sr} \tag{10}$$
$$y_{ih}^{s} + y_{jk}^{r} - 1 \leq \zeta_{ijhk}^{sr} \tag{11}$$

$\forall i, j, s, r \in V, \forall h, k \in P$, which guarantee that $\zeta_{ijhk}^{sr} = y_{ih}^{s} y_{jk}^{r}$. We call this reformulation the *usual linearization*.

The number of variables and constraints in the linearized model is $O(|V|^4|P|^2)$. Solving the formulation with a Branch-and-Bound method, we need to solve a linear relaxation at each node. Thus, it is important to obtain formulations with a small number of constraints. Since the constraint size is due to the linearization constraints (9)-(11), we propose a different linearization technique, called *compact linearization*, which reduces the size of the problem. This consists in multiplying each assignment constraints (2) by y_{jk}^{r}; the resulting bilinear terms are successively linearized by substituting them with the appropriate ζ variable: thus, we obtain valid linear relationships between the ζ and the y variables [8]. The compact linearization also assumes that $\zeta_{ijhk}^{sr} = \zeta_{jihk}^{rs}$, which holds by commutativity of product. Thus, the compact linearization for the MSPCD is as follows:

$$\forall i, j, r \in V, l \in P \qquad \sum_{h=1}^{p} \sum_{s=1}^{n} \zeta_{ijhk}^{sr} = y_{jk}^{r} \tag{12}$$
$$\forall i, j, s, r \in V, h, k \in P \qquad \zeta_{ijhk}^{sr} = \zeta_{jikh}^{rs} \tag{13}$$

Notice that although there are still $O(|V|^4|P|^2)$ constraints (13), these can be used to eliminate half of the linearizing variables and deleted from the formulation (any decent MILP pre-solver will actually perform this reformulation automatically). Hence, there are only $O(|V|^3|P|)$ constraints in the compact linearization.

2.1.2 Bounds to the objective function

Simply feeding the classic formulation to a MILP solver was found to be inefficient. To speed up the search, we artificially set lower and upper bounds to the objective function value.

Load Balancing (LB) and the Critical Path Method (CPM) are two fast alternatives to compute cheap lower bounds. The optimal solution cannot have a shorter execution time than the ideal load balancing case, i.e. when there is no idle time intervals and all the processors complete the execution exactly at the same time. In other words, if we let $T_{\max} = \max_{j \leq n}\{t_j + L_j\}$ denote the objective function, we have $T_{\max} \geq \frac{1}{p}\sum_{i=1}^{n} L_i = T_{LB}$. Furthermore, the length of final schedule can not be smaller that the length of the critical path, the longest path connecting a node with no predecessors to a node without successors. Let us denote by $\delta^+(j)$ the set of immediate successors of task j, i.e. $\delta^+(j) = \{i \in V : (j,i) \in A\}$. By using CPM the corresponding lower bound T_{CP} can be defined as $T_{CP} = \max_{j \leq n} CP(j)$ where

$$CP(j) = \begin{cases} L_j, & \text{if } \delta^+(j) = \emptyset, \\ L_j + \max\limits_{j' \in \delta^+(j)} CP(j'), & \text{otherwise,} \end{cases}$$

Let $\underline{T}$ be the greatest of the bounds obtained by LB and CPM; a constraint of the form $T_{\max} \geq \underline{T}$ is then also added to the formulation. Notice that this lower bound is valid throughout the whole solution process and does not depend on the current Branch-and-Bound node being solved.

Several efficient heuristics and meta-heuristics can provide upper bounds. These heuristics are seldom applicable to a Branch-and-Bound (B&B) algorithm because at any given (B&B) iteration, some of the variables are fixed – and it is usually difficult to force the graph-based heuristics to constrain the parts of the graph structure which correspond to the fixed variables. At the first iteration, however, no variables are fixed, which makes it possible to apply some efficient graph-based heuristics as a kind of pre-processing step to the whole (B&B) run. In our tests, we used Variable Neighborhood Search (VNS) [3] to compute tight upper bounds $\overline{T}$ to the objective function value, and a constraint $T_{\max} \leq \overline{T}$ was added to the formulation prior to starting the solution process.

2.2 Packing formulation

The idea on which this model is based is to liken task i to a rectangle of length L_i and height 1, and to pack all the rectangles representing the tasks into a larger rectangle of height $|P|$ and length W, where W is the total makespan to be minimized [6].

For each task $i \in V$ let $t_i \in \mathbf{R}$ be the starting execution time (or alternatively, the horizontal coordinate of the bottom left corner of the rectangle representing task i) and $p_i \in \mathbf{N}$ be the ID of the processor where task i is to be executed (alternatively, the vertical coordinate of the bottom left corner of the rectangle representing task i). Let W be the total makespan (alternatively, the length of the larger rectangle where we want to pack the task-rectangles). Let x_{ik} be 1 if task i is assigned to processor k, and zero otherwise. In order to enforce non-overlapping constraints, we define two sets of binary variables:

$$\forall\, i,j \in V\ \sigma_{ij} \quad = \quad \begin{cases} 1 & \text{task } i \text{ finishes before task } j \text{ starts} \\ 0 & \text{otherwise} \end{cases}$$

$$\forall\, i,j \in V\ \epsilon_{ij} \quad = \quad \begin{cases} 1 & \text{the processor index of task } i \text{ is strictly less than that of task } j \\ 0 & \text{otherwise.} \end{cases}$$

The formulation is as follows:

$$\min_{t,p,\sigma,\epsilon} \quad W \tag{14}$$

$$\forall\, i \in V \qquad t_i + L_i \leq W \tag{15}$$
$$\forall\, i \neq j \in V \qquad t_j - t_i - L_i - (\sigma_{ij} - 1)W_{\max} \geq 0 \tag{16}$$
$$\forall\, i \neq j \in V \qquad p_j - p_i - 1 - (\epsilon_{ij} - 1)|P| \geq 0 \tag{17}$$
$$\forall\, i \neq j \in V \qquad \sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \geq 1 \tag{18}$$
$$\forall\, i \neq j \in V \qquad \sigma_{ij} + \sigma_{ji} \leq 1 \tag{19}$$
$$\forall\, i \neq j \in V \qquad \epsilon_{ij} + \epsilon_{ji} \leq 1 \tag{20}$$
$$\forall\, j \in V \,:\, i \in \delta^-(j) \qquad \sigma_{ij} = 1 \tag{21}$$
$$\forall\, j \in V \,:\, i \in \delta^-(j) \qquad t_i + L_i + \sum_{h,k \in P} \gamma_{ij}^{hk} x_{ih} x_{jk} \leq t_j \tag{22}$$
$$\forall\, i \in V \qquad \sum_{k \in P} k x_{ik} = p_i \tag{23}$$
$$\forall\, i \in V \qquad \sum_{k \in P} x_{ik} = 1 \tag{24}$$
$$W \geq 0 \tag{25}$$
$$\forall\, i \in V \qquad t_i \geq 0 \tag{26}$$
$$\forall\, i \in V \qquad p_i \in \{1, \ldots, |P|\} \tag{27}$$
$$\forall\, i \in V, k \in P \qquad x_{ik} \in \{0, 1\} \tag{28}$$
$$\forall\, i, j \in V \qquad \sigma_{ij}, \epsilon_{ij} \in \{0, 1\}, \tag{29}$$
$$\tag{30}$$

where $W_{\max}$ is an upper bound on the makespan W, e.g.

$$W_{\max} = \sum_{i \in V} L_i + \sum_{i,j \in V} c_{ij} \max\{d_{hk} \mid h, k \in P\}.$$

We minimize the makespan in (14) and (15), we define the time order on the tasks in terms of the σ variables in (16), and similarly we define the CPU ID order on the tasks in terms of the ϵ variables in (17). By (18), the rectangles must have at least one relative positioning on the plane. By (19) a task cannot both be before and after another task; similarly, by (20) a task cannot be placed both on a higher and on a lower CPU ID than another task. Constraints (21) enforce the task precedences; constraints (22) model the communication delays. Constraints (23) link the assignment variables x with the CPU ID variables p, and (24) are the assignment constraints.

Formulation (14)-(29) is also a bilinear MINLP where the bilinearities arise because of the communication delay constraints. Observe, however, that this formulation has variables with only two indices, as opposed to the formulation of Section 2.1 whose variables have three indices. Since linearization squares the size of the variable set, it appears clear that the packing formulation size is much smaller than the classic one.

2.2.1 Linearization of the packing formulation

We linearize formulation (14)-(29) by introducing variables $0 \leq z \leq 1$ as follows:

$$\forall\, j \in V, i \in \delta^-(j), h, k \in P\ (z_{ij}^{hk} = x_{ih} x_{jk}).$$

Integrality on the linearizing variables z follows from the usual linearization constraints:

$$\forall\, j \in V, i \in \delta^-(j), h, k \in P\ (x_{ih} \geq z_{ij}^{hk} \wedge x_{jk} \geq z_{ij}^{hk} \wedge x_{ih} + x_{jk} - 1 \leq z_{ij}^{hk}) \tag{31}$$

The above linearization constraints are equivalent to the following compact linearization:

$$\forall\, i \neq j \in V, k \in P\ (\sum_{h \in P} z_{ij}^{hk} \quad = \quad x_{jk}). \tag{32}$$

$$\forall\, i \neq j \in V, h, k \in P\ (z_{ij}^{hk} \quad = \quad z_{ji}^{kh}). \tag{33}$$

Although the set of compact linearization constraints (33)-(32) is smaller than the set of usual linearization constraints (31), the difference is not as dramatic as for the classic formulation ($O(|V|^2|P|)$ against $O(|V|^2|P|^2)$). Experimentally, we found that the compact linearization is beneficial when the precedence graph is dense. For sparse graphs the usual linearization sometimes yields shorter solution times. This is in accordance with the observation that a more precise evaluation of the magnitude order of the usual linearization constraint size is $O(|A||P|^2)$ rather than $O(|V|^2|P|^2)$, and that $|A|$ decreases with respect to $|V|^2$ as the sparsity of the graph increases.

3 Computational results

In this section we describe and discuss the computational experiments performed to test the ideas presented in this paper. All our computations were carried out on an Intel PIV 2.66GHz CPU with 1GB RAM running Linux. We used CPLEX v. 8.1 [7].

3.1 Instances

Several small size test instances known from the literature (like [11]) were used in our experiments. In addition, we tested a few examples with known optimal solutions (named with the prefix `ogra`) generated as in [2]. We scheduled small-sizes instances on: $p = 2$ and $p = 3$, completely connected processors. For `ogra_` examples the optimal solution is obtained when the tasks are well packed (as in ideal schedule) in the order defined by the task indices. For these task graphs it is always the case that $T_{max} = T_{LB} = T_{CP}$. This means that they have a special structure which makes them very hard instances for heuristic methods. In other words, in the cases when the task graph density is small, i.e. the number of mutually independent tasks is large, it is hard to find the task ordering which yields the optimal schedule.

Medium-size instances (with 30 and 40 tasks) are generated randomly, as it was proposed in [2]. These instances names are generated with the prefix `t` followed by the number of tasks, density and index distinguishing graphs with the same characteristics. They are scheduled on a ring of $p = 4$ processors.

3.2 Resulting tables

Here we present some selected computational results. In Table 1 comparative results for both formulations are given for small test instances. First four columns contain the task graph characteristics (name, number of processors, number of tasks, edge densities). The length of the optimal schedule is given in the fifth column, while the last two columns contain the time required by CPLEX to solve classical and packing formulation respectively. The time is represented by <hours>:<minutes>:<seconds>.<hundreds of seconds> with zero values in front omitted.

As we can see, with all improvements (artificial lower bounds, VNS heuristic initial solution) we were not able to solve to the optimality graphs with more that 10 tasks using classical formulation. On the other hand, with packing formulation 20 tasks represent no significant difficulty.

Medium-size examples seem to be difficult even for packing formulation. Results given in the Table 2 show that more that 24 hours are needed for most of them to be solved to the optimality.

Table 1: Results for small task graph examples

example	p	n	ρ	T_{max}	Classic	Packing
ogra_1	3	9	13.89	20	2:14:51	0.41
ogra_2	3	9	27.78	20	34:31	0.22
ogra_3	3	9	41.67	20	16:34	0.13
mt91	2	10	31.11	53	46:05	0.08
ogra_4	3	10	13.33	25	15:42:27	30.72
ogra_5	3	10	26.67	25	1:13:14	0.20
ogra_6	3	10	40.00	25	1:12:31	0.18
gra12	3	12	16.67	135	—	20:48.92
ogra_7	4	12	20	50	—	0.67
t20_10	4	20	8.95	110	—	45:43:51.52
t20_25	4	20	23.68	151	—	29:29.86
t20_50	4	20	53.16	221	—	17.06
t20_75	4	20	76.32	241	—	51:56.79
t20_90	4	20	80.52	242	—	11.72

Namely, for these examples we set time limit of 24 hours for the CPLEX execution. In the fourth column of Table 2 we put the required execution time to obtain the optimal solution (if it has been reached) or the given time limit (if the optimal solution remind unfound). The last columns contains the gap between the final solution and the obtained lower bound.

Table 2: Results for medium-size task graph examples

example	p	T^*_{max}	Time	Gap
t30_50_2	4	470	24:00:00	7.12%
t30_60_1	4	467	06:44:23	0.00%
t30_60_2	4	451	24:00:00	5.31%
t30_75_1	4	544	23:11:21	0.00%
t40_10_1	4	233	24:00:00	2.15%
t40_10_2	4	245	24:00:00	25.31%
t40_25_1	4	270	01:24:08	0.00%
t40_25_2	4	369	24:00:00	24.39%
t40_30_1	4	457	24:00:00	7.98%

The average CPU time improvement of the packing formulation over the best variant of the classic formulation is experimentally proved to be about 5000 times. Therefore, we believe that the packing formulation represents a valuable tool for solving small MSPCD instances to optimality.

4 Conclusion

In this paper two linear programming formulations for scheduling dependent tasks onto homogeneous multiprocessor architecture of an arbitrary structure with taking into account communication delays is proposed. The models are used within the commercial software for solving Mixed-Integer

Linear Programs CPLEX. To reduce the execution time required by CPLEX to solve test examples to the optimality, we added heuristic limitations to the upper bound and the lower bound and introduce reduction constraints to the original model.

References

[1] E. H. Bowman (1959), The schedule-sequencing problem, *Operations Research* **7**(5), 621 – 624.

[2] T. Davidović and T. G. Crainic (2006), Benchmark problem instances for static task scheduling of task graphs with communication delays on homogeneous multiprocessor systems, *Computers & OR* **33**(8), 2155 – 2177.

[3] T. Davidović, P. Hansen, and N. Mladenović (2005), Permutation based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communication delays, *Asia-pacific Journal of Operational Research* **22**(3), 297 – 326.

[4] R. Fortet (1960), Applications de l'algèbre de boole en recherche opérationelle, *Revue Française de Recherche Opérationelle* **4**, 17 – 26.

[5] P.L. Hammer and S. Rudeanu (1968), *Boolean Methods in Operations Research and Related Areas*, Springer, Berlin.

[6] Y. Guan and R.K. Cheung (2004), The berth allocation problem: models and solution methods, *OR Spectrum* **26**(1), 75 – 92.

[7] ILOG (2002), *ILOG CPLEX 8.0 User's Manual*, ILOG S.A., Gentilly, France.

[8] L. Liberti (2005), Linearity embedded in nonconvex programs, *Journal of Global Optimization* **33**(2), 157 – 196.

[9] N. Maculan, C. C. Ribeiro, S.C.S. Porto, and C. C. de Souza (1999), A new formulation for scheduling unrelated processors under precedence constraints, *RAIRO Recherche Operationelle* **33**, 87 – 90.

[10] A. S. Manne (1960), On the job–shop scheduling problem, *Operations Research* **8**(2), 219 – 223.

[11] S. Manoharan and P. Thanisch (1991), Assigning dependency graphs onto processor networks, *Parallel Computing* **17**, 63 – 73.

[12] M. Queyranne and A. Schulz (1994), Polyhedral approaches to machine scheduling, *Technical report*, **TUB:1994–408**, Technical University of Berlin.

[13] G. C. Sih and E. A. Lee (1993), A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, *IEEE Trans. Parallel and Distributed Systems* **4**(2), 175 – 187.

[14] T. Tobita and H. Kasahara (2002), A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *Journal of Scheduling* **5**(5), 379 – 394.

The Cost of Scheduling Customers in Routing Problems

Wout Dullaert
Institute of Transport and Maritime Management Antwerp,
University of Antwerp, Keizerstraat 64, 2000 Antwerp, Belgium, wout.dullaert@ua.ac.be
Olli Bräysy
Agora Innoroad Laboratory, Agora Center,P.O. Box 35, FI-40014 University of Jyväskylä, Finland,, Olli.Braysy@jyu.fi
Amir Salehipour
Department of Applied Economics, University of Antwerp, Antwerp, Belgium, amir.salehipour@ua.ac.be

The cost of servicing customers in routing problems tends to differ according to the characteristics of their service. In this paper, several methods are developed for determining the incremental cost of a customer in a heterogeneous routing problem with time windows and identifying its cost drivers.

Keywords: vehicle routing, incremental costs, scheduling

1. Introduction

Both for less-than-truckload (LTL) and full-truckload (FTL) routing and scheduling, customers tend to differ a lot with respect to the flexibility they give to the dispatcher to design routes. The more flexibility they give, the more cost efficient the routes the dispatcher can design. In practice, the cost-driving effect of scheduling inflexibility has been recognized for a longer time. E.g., express delivery companies charge higher rates for customers requiring a faster delivery. In most cases, reliable cost estimates of the various sources of scheduling inflexibility are unavailable.
Correct estimates of the incremental cost of customers or customer types are nevertheless essential for pricing routing services and accepting new customers. The additional cost of routing a customer acts as a price-floor: the freight rate should at least cover the additional cost of servicing that customer. This information remains valuable, even if prices in the industry are not cost-based. If prices are aimed at capturing customers' willingness to pay (e.g., perceived value pricing) or if prices are based on competitors' price levels (e.g., going-rate pricing), the incremental cost of servicing a customer is vital for determining the contribution or profitability of servicing each customer.
In this paper, several methods for determining the incremental cost of a customer in a vehicle routing problem with heterogeneous vehicles in which service at customers is restricted to prespecified hard time windows. The Fleet Size and Mix Vehicle Routing Problem (FSMVRP) is a Vehicle Routing Problem (VRP, see Toth and Vigo [17]) where the homogeneous fleet assumption of the traditional VRP has been lifted. A number of vehicle types with different capacities and acquisition costs are given. The objective is to find a fleet composition and a corresponding routing plan that minimizes the sum of routing and vehicle costs. For reviews on the FSMVRP, we refer the reader to Salhi and Rand [14], Osman and Salhi [13], and Lee et al. [10].
Section 2 presents 8 different approximation methods for estimating the incremental cost of a customer. They are extensively tested on problem instances for heterogeneous routing problems in Section 3. This section also looks at factors influencing the incremental costs of customers. Section 4 concludes the paper.

2. Approximations for incremental costs in routing

To determine the exact incremental cost of routing a customer two routing problems have to be solved to optimality: once with and once without the customer involved. The difference yields the additional cost generated by that customer. Exact solution algorithms for routing and scheduling

are only capable of solving relatively small problem instances (Cordeau et al. [8]). For real-life applications, one must resort to heuristic procedures to approximate the optimal solution (without quality guarantee) within a reasonable amount of computation time. For the determination of a customer's incremental cost, these computation time considerations are even more an issue given the large number of computations involved.

Re-solving an entire routing problem for each customer is a computationally expensive procedure to determine the incremental routing costs. For a small routing problem of 100 customers, 100 routing problems of 99 customers have to be solved to determine the incremental cost of each customer. In real-life applications, the number of customers to be routed daily is a lot larger. For strategic decisions, company activity probably needs to be simulated of a large number of days to obtain reliable cost estimates. The number of routing problems that needs to be solved to obtain the incremental cost of different categories of customers soon becomes unmanageable, even for the fast, high-quality metaheuristics available today. Therefore more efficient ways of estimating the incremental routing cost of customers are suggested in this paper.

In this paper, we evaluate different approaches for determining the incremental cost. Full and partial re-optimization approaches are compared to naïve re-optimization procedures that simply remove a customer from the route without changing the sequence of the other customers in the route. Partial re-optimization procedures, reschedule only a part of the original routing problem. Clearly, the route containing the customer under consideration needs to be rescheduled as the customer is temporarily removed. But also the customers of a number of adjacent routes should be included in the analysis. Once the number of routes to be rescheduled is determined, local search heuristics can be used to re-optimize the routes. The re-optimization procedure that we applied here is the same as for the full re-optimization process.

The routing costs of the initial routing problem with all customers involved acts as a benchmark for all cost approximations. It should therefore be solved with the best possible solution method available. The recently developed multi-start deterministic annealing (MSDA) heuristic for the FSMVRPTW (Bräysy et al., [7]) obtained 167 best known solutions (of which 165 new best solutions) for the 168 problem instances from the literature. The heuristic is also time efficient, making it well-suited to calculate the incremental cost of each customer. The metaheuristic comprises three phases. In the first phase high quality initial solutions are generated by means of a savings-based heuristic combining diversification strategies with learning mechanisms. In phase two an attempt is made to reduce the number of routes in the initial solution with a new local search procedure and in phase three the solution from phase two is further improved by a set of four local search operators that are embedded in a deterministic annealing framework to guide the improvement process. For more details, the reader is referred to Bräysy et al. [7].

From a managerial point of view vehicle costs can be ignored in the so-called 'short run', when capacity cannot be adjusted. In the long run, by definition, all inputs of the production process are variable and therefore vehicle costs should be part of the objective function. The heuristic of Bräysy et al. [7] was adapted to be able to minimize both objective functions. By using a number of new implementation strategies and efficient (time window) feasibility checks, computation time is small compared to the solution quality obtained. We studied the following 8 metrics:

(1) Full re-optimization: In full re-optimization each customer is removed from the set of customers and a new solution is build from scratch. The total routing costs of the full set minus the total routing costs of the reduced set of customers, probably yields the best possible cost estimate of the incremental cost of that particular customer.

(2) DA500 and (3) DA100: For DA500 and DA100, the structure of the initial routing problem is maintained after removing the customer under consideration. After removing the customer, the deterministic annealing phase of the MSDA heuristic (phase 3) is executed for respectively 500 or 100 iterations.

(4) Local optimization: In the original implementation of the final improvement stage of the MSDA heuristic (phase 3), a threshold is used to allow for a controlled escape from local optima.

For the local minimum cost estimate, this threshold is not used as the search ends as soon as a local optimum is found. Except for this modification, the local minimum cost approximation is identical to the original improvement phase of the MSDA heuristic for the FSMVRPTW.

(5) Single route optimization: After removing a customer, Single Route Optimization checks whether the route can be serviced by a smaller vehicle. If so, possible savings in distance and vehicle cost are recorded. Then IOPT is applied to this tour only and additional distance savings are recorded. The IOPT intra-tour operator is a generalization of Or-opt (Or [12]). It considers segments of any length and also includes moves where the segment is reversed before it is relocated (Bräysy [1]). The reported value is the sum of the savings generated by the possible use of a smaller vehicle and the distances savings obtained by IOPT.

(6) Close re-optimization I: This approximation method is similar to the local optimization method (4) except the fact that we look for improvements within the route from which the customer was removed and its neighboring routes only within a distance limit that is adjusted during the search. As described in Bräysy et al. [7], the distance limit for the initial solution is set to the distance of the farthest customer from the depot to represent some kind of potential maximum arc length in a solution, multiplied by a uniform number r (between 0 and 1). During the search information is gathered on the maximum distances that have still made improvements possible. The maximum distance that has still enabled improvement is used as the new reference and multiplied by *r* to get the new distance limit.
For Close Re-optimization I, only customers within the current distance limit are taken into consideration. Of the four possible local search operators from stage 3 of the MSDA heuristic, the route splitter is not used and route elimination is only used in the beginning of the search. The search is mainly based on the IOPT and ICROSS operators. Cross-exchanges (Taillard et al. [16]) relocate or exchange segments of consecutive customers from different routes while preserving orientation. The ICROSS neighborhood (Bräysy et al. [2]) used here extends the neighborhood by also including moves where the current segment is reversed before insertion and allowing for adjustments of vehicle types.
To increase the efficiency of the improvement phase, ICROSS and IOPT do not consider a route pair or a single route if no improvement was found last time and the routes have not changed since. So if customer 1 was removed from route 1, and there are e.g., 3 routes close to it (e.g., 3,4,5) , only route pairs (1,3), (1,4) and (1,5) are evaluated by ICROSS and IOPT.

(7) Close Re-optimization II: Close Re-optimization II is similar to Close Re-optimization I. The only difference is that set of routes considered for re-optimization is adjusted during the search. In the first stage, only routes that are located within the distance limit are considered for re-optimization. In the next stage, the set is extended by routes that are closed to routes for which an improvement was obtained in the previous step. This process is repeated for as long as new improvements can be found in the newly defined sets of routes. So e.g., in the above example, if an improvement was found for the route pair (1,3), also combinations with routes close to route 3 will be considered etc.
(8) Simple removal: Simple removal is clearly the most naïve cost estimate. It simply consists of re-linking the current route after removing the customer under consideration, without changing the sequence of the customers. Only cost differences related to distance savings are recorded.

3. Computational testing

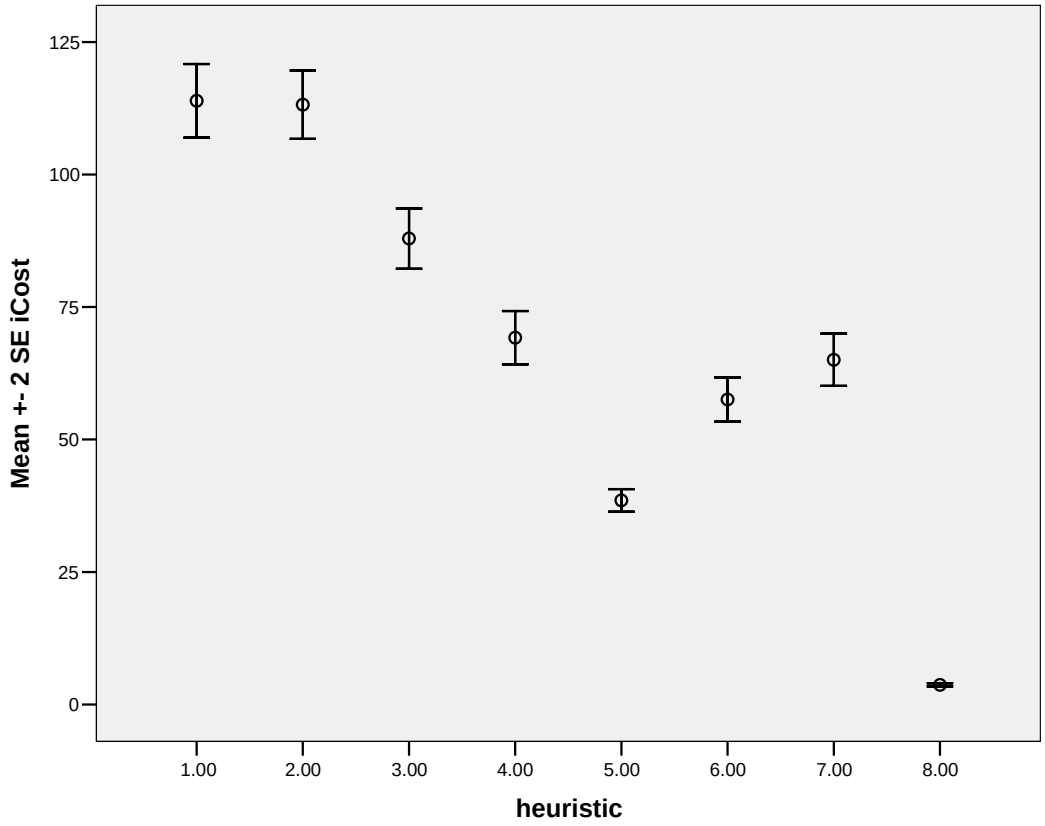
The cost approximations were implemented in Java (JDK 5.0) and tested on an AMD Athlon 2600+ (512 MB RAM) computer. The computational experiments were performed on the benchmark instances proposed by Liu and Shen [11], derived from the well-known VRPTW instances of Solomon [15]. Solomon's problem sets for the VRPTW consist of 56 instances of 100 customers with randomly generated coordinates (set R), clustered coordinates (set C) or both (semi-clustered RC set). Subsets R1, C1 and RC1 have a short scheduling horizon and small vehicle capacities. By

contrast, subsets R2, C2 and RC2 may have routes with a larger number of customers. For each six subsets Liu and Shen introduced several vehicle types with different capacities and costs. In addition, three different vehicle cost structures A, B and C were suggested, resulting in total to 168 problem instances. Here cost structure A refers to largest vehicle costs and C to the smallest.

3.1. Long-run incremental costs

Due to space limitations, we here only report on long-run incremental cost estimates. In the long-run, capacity can be adjusted and as a consequence, both vehicle and distance related costs should be reflected in the incremental cost estimate of a customer. Of the three different cost structures for vehicles used in Liu and Shen [11], we consider cost structures A (largest vehicle costs) and C (smallest vehicle costs). The same 6 problem instances are used to test the cost approximation methods, resulting in a test bed of 1200 customers for which incremental costs have to be estimated.

Figure 1: Long-run cost estimates



Full re-optimization (1) and DA500 (2) give the highest estimates for the incremental cost of a customer. Average performance is clearly lower from the third approach (DA100) onwards. As for the estimation of short run incremental costs, Figure 1 seems to suggest that the variance is not equal for the different cost estimators. This is confirmed by the standard deviation and standard error statistics in Table 6 and the formal Levene test statistic (289.466, Sig 0.000) rejecting the null hypothesis that the group variances are equal.

Given that the groups are of equal size, ANOVA is used to reject the hypothesis that average incremental cost estimates are equal across heuristics used (F = 227.578, Sig. 0.000). Using the post

hoc Tamhane T2 test (Table 2) the following differences between group means for the heuristics were found to be significant at the 5% level.

Table 1: Multiple comparisons for long run costs (A and C) based on Tamhane T2

	1	2	3	4	5	6	7	8
1		-0.7209535	-25.981485	-44.695939	-75.398092	-56.347825	-48.861196	-110.20222
2	0.7209535		-25.260531	-43.974986	-74.677138	-55.626871	-48.140243	-109.48126
3	25.981485	25.260531		-18.714455	-49.416607	-30.36634	-22.879711	-84.220731
4	44.695939	43.974986	18.714455		-30.702152	-11.651885	-4.1652568	-65.506277
5	75.398092	74.677138	49.416607	30.702152		19.050267	26.536896	-34.804124
6	56.347825	55.626871	30.36634	11.651885	-19.050267		7.4866283	-53.854392
7	48.861196	48.140243	22.879711	4.1652568	-26.536896	-7.4866283		-61.34102
8	110.20222	109.48126	84.220731	65.506277	34.804124	53.854392	61.34102	

Table 1 indicates that the full optimization (1) and DA500 (2) cost approximation have a similar performance. The group means of DA100 (3), single route optimization (5) and simple removal 8) are statistically different from all other cost estimators. The difference between local minimum (4) and close route optimization II (7) was not considered to be significant. The same goes for the difference between close route optimization I (6) and II (7), the difference between local minimum (4) and close route optimization I (6) being different at the 5% significance level.

For large datasets, statistical tests often have the drawback that null hypotheses are too easily rejected and due to the large number of observations practically non-significant variables are declared as statistically significant. This and the fact that the ability to save on vehicle costs matters most when vehicle costs are significant as they are in practice, we also report results on the A cost structure only. For the customers in the six problem instances for cost structure A, the null hypothesis of equal variances was rejected (Levene statistic 322.541, Sig 0.000). Because the equality of means across groups was rejected (F = 160.765, Sig 0.000), Table 2 list the differences between group means for the heuristics identified by the Tamhane T2 test at the 5% level. Contrary to the previous case, the difference between the performance of close re-optimization I and II is no longer considered to be significant.

Table 2: Multiple comparisons for long run costs (cost structure A) based on Tamhane T2

	1	2	3	4	5	6	7	8
1		-9.781975	-44.267551	-73.090543	-125.78218	-91.385472	-78.344565	-160.96156
2	9.781975		-34.485576	-63.308568	-116.00021	-81.603497	-68.56259	-151.17958
3	44.267551	34.485576		-28.822993	-81.51463	-47.117921	-34.077015	-116.69401
4	73.090543	63.308568	28.822993		-52.691637	-18.294928	-5.2540218	-87.871015
5	125.78218	116.00021	81.51463	52.691637		34.396709	47.437615	-35.179378
6	91.385472	81.603497	47.117921	18.294928	-34.396709		13.040906	-69.576086
7	78.344565	68.56259	34.077015	5.2540218	-47.437615	-13.040906		-82.616993
8	160.96156	151.17958	116.69401	87.871015	35.179378	69.576086	82.616993	

3.2. Cost drivers and approximating cost estimators

Table 3 illustrates that DA500 provides the best cost estimates if one ignores full re-optimization as a viable approach to determining incremental costs.

Table 3: Hit rates and computing times for short and long costs

	short run costs			Long run costs (cost structure A)		
	overall best	best of 7	total CPU	overall best	best of 7	total CPU
1	0.5533		176.32	0.6117		341.70
2	0.4533	0.9917	92.95	0.3883	0.9983	196.43
3	0.0467	0.1617	18.59	0.0317	0.1150	39.10
4	0.0100	0.0650	1.14	0.0083	0.0433	3.03
5	0.0033	0.0200	0.01418	0.0000	0.0017	0.01457
6	0.0050	0.0317	0.44900	0.0033	0.0150	0.63700
7	0.0033	0.0350	0.48500	0.0083	0.0417	1.54500
8	0.0000	0.0050	0.00093	0.0000	0.0017	0.00162

Both for the short and long run costs (cost structure A), DA500 generates the best possible cost estimates in more than 99% of the cases and within acceptable computation times (99.17% and 99.83% respectively).

Although high quality cost estimators have significant business value, logistics practitioners will also be interested in unraveling the underlying cost structure. By identify the main cost drivers of servicing customers, the cost of servicing a (new) customer could be easily estimated based on the customers characteristics without having to calculate any specific cost estimate. Moreover, information on cost drivers could be used to design price tariffs.

Eight different cost drivers were therefore considered for explaining the variation in the short-run cost estimates of DA500: the size of customer demand, the distance from the depot, the width of the service time window and the number of customers located within 5, 10, 20, 30 and 50% of the maximum distance in the problem. Stepwise linear regression (entry probability 5%, removal probability 10%) was used to explain the variation in the DA500 cost estimates. The models were unable to explain more than 50% of the variation in the DA500 incremental costs estimates and its parameter estimates sometimes had wrong signs. Explaining or predicting incremental costs based on cost drivers therefore seems to be of little value, although this conclusion is based on theoretical problem instances only. It may well be that this approach proves useful in real-life routing problems.

Table 3 illustrates that computation times are significantly higher for the first two incremental cost estimators (CPU times in seconds for all customers in the problem set). It would therefore be beneficial if one could obtain equally good cost estimates by identifying a strong relationship between the cost estimates obtained by the most powerful and time consuming cost approximation methods and the faster ones. Using stepwise regression (entry probability 5%, removal probability 10%) the following models were obtained for the dependent variable DA500 for estimating short cost run costs (see Tables 4 and 5).

Table 4: Model coefficients

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig	Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
1	(Constant)	18.000	1.298		13.871	.000					
	Single_route	1.330	.025	.912	54.266	.000	.912	.912	.912	1.000	1.000
2	(Constant)	14.619	1.413		10.348	.000					
	Single_route	.777	.105	.533	7.432	.000	.912	.291	.122	.052	19.089

close_route1	.602	.111	.389	5.424	.000	.908	.217	.089	.052	19.089

Table 5: Model summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.912(a)	.831	.831	21.96072	.831	2944.792	1	598	.000
2	.916(b)	.839	.839	21.45679	.008	29.418	1	597	.000

a Predictors: (Constant), single_route
b Predictors: (Constant), single_route, close_route1

Although explaining the variation in the cost estimates obtained by DA500 by means of a constant and the cost estimates of Single Route Optimization and Close Re-optimization I (model 2), obtained a higher adjusted R square, it cannot be preferred over Model 1. Model 2 suffers from a high degree of collinearity (Tolerance level < 0.10) due to the fact that the two explaining variables are highly correlated. The regression equation DA500 = 18.00 + 1.330 x cost estimate of Single Route Optimization (5) contains highly significant (from zero) coefficients and is capable of explaining 83.10% of the variation in the cost estimates of DA500, which can make it an interesting tool for on average obtaining higher quality cost estimates within a fraction of the computation cost (0.01418 versus 92.95). This difference is even larger when modeling the long run costs. For the DA500 long run cost estimates for cost structure A and the Single Route Optimization cost estimator the following model was obtained : DA500 = 31.705 + 3.315 x Single Route Optimization. Both explanatory variables are significantly different from zero (t-values of 9.002 and 50.274 respectively) and the model has an adjusted R square of 0.808.

4. Conclusion

This paper offers eight different approximations for the incremental cost of a customer. Full and partial re-optimization approaches are compared to naïve re-optimization procedures that simply remove a customer from the route without changing the sequence of the other customers in the route. The approaches clearly differ with respect to solution quality and CPU time requirements. Both for short and long run costs, DA500, an approximation based on the FSMVRPTW heuristic from Bräysy et al. [7] was considered to be the most powerful approach next to a time consuming full re-optimization approach.
Computation testing on theoretical benchmarks indicated that even a wide set of possible cost drivers cannot sufficiently model the incremental cost of a customer. Less than 50% of the variation in incremental cost estimates could be explained by cost drivers related to time window size, customer proximity and demand size. We therefore believe that approximating incremental costs either by DA500 or by a regression equation based on even faster cost estimators (e.g., Single Route Optimization) is a more fruitful approach to estimate incremental costs in real-life routing problems.

Acknowledgement

This work was partially supported by the EDGE project of the Research Council of Norway, the KAKS Foundation and the Research Foundation - Flanders (Belgium). This support is gratefully acknowledged. The authors would also like to thank Bruno De Borger, Peter Goos and Piet Rietveld for their constructive comments on an earlier version of this paper. The usual disclaimer applies.

References

[1] O. Bräysy (2003), A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows, *INFORMS Journal on Computing* **15**(4), 347 – 368.

[2] O. Bräysy, G. Hasle and W. Dullaert (2004), A multi-start local search algorithm for the vehicle routing problem with time windows, *European Journal of Operational Research* **159**, 586 – 605.

[3] O. Bräysy and M. Gendreau (2005), Vehicle routing problem with time windows, Part I: route construction and local search algorithms, *Transportation Science* **39**, 104 – 118.

[4] O. Bräysy and M. Gendreau (2005), Vehicle routing problem with time windows, Part II: metaheuristics, *Transportation Science* **39**, 119 – 139.

[5] O. Bräysy, M. Gendreau, G. Hasle and A. Løkketangen (2006a), A survey of heuristics for the vehicle routing problem, part I: basic problems and supply side extensions, Working paper, SINTEF ICT, Norway.

[6] O. Bräysy, M. Gendreau, G. Hasle and A. Løkketangen (2006b), A survey of heuristics for the vehicle routing problem, Part II: Demand Side Extensions, Working paper, SINTEF ICT, Norway.

[7] O. Bräysy, W. Dullaert, G. Hasle, D. Mester and M. Gendreau (2007), An Effective Multi-restart Deterministic Annealing Metaheuristic for the Fleet Size and Mix Vehicle Routing Problem with Time Windows, Paper submitted to Transportation Science.

[8] J. F. Cordeau, G. Laporte and A. Mercier (2001), A unified tabu search heuristic for vehicle routing problems with time windows, *Journal of the Operational Research Society*, **52**, 928 – 936.

[9] W. Dullaert, O. Bräysy, and G. Van de Weyer (2005), Scheduling flexibility in vehicle routing. In: Witlox, F., Dullaert, W. and B. Vernimmen (Eds.). *Proceedings of the BIVEC-GIBET Research Day*, Part I, 339 – 350.

[10] Y. H. Lee, Kim J. I., Kang K. H. and Kim K. H. (2006), A heuristic for vehicle fleet mix problem using tabu search and set partitioning, Working paper, Yonsei University, Seoul, Korea.

[11] F. H. Liu and S. Y. Shen (1999), The fleet size and mix vehicle routing problem with time windows, *Journal of the Operational Research Society* **50**, 721-732.

[12] I. Or (1976), Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. Ph.D. thesis, Northwestern University, Evanston, IL.

[12] I. H. Osman and Salhi S. (1996), Local search strategies for the vehicle fleet mix problem. V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, G.D. Smith, eds. *Modern heuristic search methods*. Chichester, John Wiley & Sons, 131 – 153.

[13] S. Salhi and G. Rand (1993), Incorporating vehicle routing into the vehicle fleet composition problem, *European Journal of Operational Research* **66**, 313 – 360.

[14] M. M. Solomon (1987), Algorithms for the vehicle routing and scheduling problems with time window constraints, *Operations Research*, **35** 254 – 265.

[15] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J. Y. Potvin (1997), A tabu search heuristic for the vehicle routing problem with soft time windows, *Transportation Science* **31**, 170 – 186.

[16] P. Toth and D. Vigo (Eds) (2001), *The Vehicle Routing Problem.* SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia, USA.

Time-Indexed Formulations and a Large Neighborhood Search for the Resource-Constrained Modulo Scheduling Problem

Benoît Dupont de Dinechin
STMicroelectronics STS/CEC, 12, rue Jules Horowitz - BP 217. F-38019 Grenoble,
benoit.dupont-de-dinechin@st.com

The resource-constrained modulo scheduling problem is motivated by the 1-periodic cyclic instruction scheduling problems that are solved by compilers when optimizing inner loops for instruction-level parallel processors. In production compilers, modulo schedules are computed by heuristics, because even the most efficient integer programming formulation of resource-constrained modulo scheduling by Eichenberger and Davidson appears too expensive to solve relevant problems.

We present a new time-indexed integer programming formulation for the resource-constrained modulo scheduling problem and we propose a large neighborhood search heuristic to make it tractable. Based on experimental data from a production compiler, we show that this combination enables to solve near optimally resource-constrained modulo scheduling problems of significant size. We also show that our large neighborhood search benefits to a lesser extent the resource-constrained modulo scheduling integer programming formulation of Eichenberger and Davidson.

Keywords: Cyclic Scheduling, Modulo Scheduling, Instruction Scheduling, Time-Indexed Formulation, Large Neighborhood Search, Adaptive Margins

1 Introduction

Modulo scheduling is the cyclic instruction scheduling framework used by highly optimizing compilers to schedule instructions of innermost program loops [1]. Modulo scheduling is an effective optimization for instruction-level parallel processors [15], especially the Very Long Instruction Word (VLIW) processors that are used for media processing in embedded devices such as set-top boxes, mobile phones, and DVD players. An example of a modern VLIW architecture is the Lx [9], which provides the basis of the successful STMicroelectronics ST200 VLIW processor family.

The modulo scheduling framework is distinguished by its focus on 1-periodic cyclic schedules with integral period, which leads to simplifications compared to the classic formulation of cyclic scheduling [10]. In the modulo scheduling framework, the period is called *initiation interval* and is the main indicator of the schedule quality. A *resource-constrained modulo scheduling problem* (RCMSP) is a modulo scheduling problem where the resource constraints are adapted from the renewable resources of the resource-constrained project scheduling problem [2].

Optimal solutions to the resource-constrained modulo scheduling problem can be obtained by solving the classic integer programming formulations by Eichenberger and Davidson [8]. However, solving such formulation is only tractable for modulo scheduling problems that comprise less than several tenth of operations[1]. While developing the ST200 production compiler at STMicroelectronics [6], we found that modulo scheduling heuristics appeared to loose effectiveness beyond such problem sizes, according to the lower bounds on the period obtained by relaxations.

In order to build high-quality modulo schedules for instruction scheduling problems of significant size, our contributions are as follows: we show that for any assumed period λ, the RCMSP appears

[1] An operation is an instance of an instruction in a program text. An instruction is a member of the processor instruction set architecture (ISA).

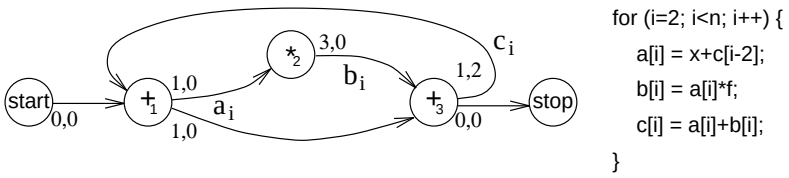


Figure 1: Sample cyclic instruction scheduling problem.

as a resource-constrained project scheduling problem with maximum time lags (RCPSP/max) and modulo resource constraints (Section 2); we present a new time-indexed integer programming formulation for the RCMSP, by adapting the time-indexed integer programming of Pritsker et al. [13] for the RCPSP/max (Section 3); we propose a large neighborhood search (LNS) heuristic for the RCMSP, based on the adaptive reduction of margins and the resolution of the resulting time-indexed integer programming formulations by implicit enumeration (Section 4).

2 The Resource-Constrained Modulo Scheduling Problem

2.1 Resource-Constrained Cyclic Scheduling Problems

A *basic cyclic scheduling problem* [10] considers a set of generic operations $\{O_i\}_{1\leq i\leq n}$ to be executed repeatedly, thus defining a set of operation instances $\{O_i^k\}_{1\leq i\leq n}^{k>0}, k \in \mathbb{N}$. We call *iteration* k the set of operation instances $\{O_i^k\}_{1\leq i\leq n}$. For any $i \in [1,n]$ and $k > 0 \in \mathbb{N}$, let σ_i^k denote the schedule date of operation instance O_i^k. Basic cyclic scheduling problems are constrained by *uniform dependences* denoted $O_i \overset{\theta_i^j,\omega_i^j}{\longrightarrow} O_j$, where the *latency* θ_i^j and the *distance* ω_i^j are non-negative integers:

$$O_i \overset{\theta_i^j,\omega_i^j}{\longrightarrow} O_j \Longrightarrow \sigma_i^k + \theta_i^j \leq \sigma_j^{k+\omega_i^j} \quad \forall k > 0$$

In this work, we are interested in *resource-constrained cyclic scheduling problems*, whose resource constraints are adapted from the *resource-constrained project scheduling problems* (RCPSP) [2]. Precisely, we assume a set of *renewable resources*, also known as *cumulative resources*, whose availabilities are given by an integral vector $\vec{B}$. Each generic operation O_i requires $\vec{b}_i$ resources for p_i consecutive units of time and this defines the resource requirements of all the operation instances $\{O_i^k\}^{k>0}$. For the cyclic scheduling problems we consider, the cumulative use of the resources by the operation instances executing at any given time must not exceed $\vec{B}$.

To illustrate the resource-constrained cyclic scheduling problems that arise from instruction scheduling, consider the inner loop source code and its dependence graph displayed in Figure 1. To simplify the presentation, we did not include the memory access operations. Operation O_3 (c[i]=a[i]+b[i]) of iteration i must execute before operation O_1 (a[i]=x+c[i-2]) of iteration i+2 and this creates the uniform dependence with distance 2 between O_3 and O_1 (arc c_i in Figure 1). The dummy operations here labeled start and stop are introduced as source and sink of the dependence graph but have no resource requirements.

Assume this code is compiled for a microprocessor whose resources are an adder and a multiplier. The adder and the multiplier may start a new operation every unit of time. However, due to pipelined implementation, the multiplier result is only available after 3 time units. This resource-constrained cyclic scheduling problem is defined by $p_1 = p_2 = p_3 = 1$, $\vec{B} = (1,1)^t$, $\vec{b}_1 = \vec{b}_3 = (0,1)^t$, $\vec{b}_2 = (1,0)^t$. The dependences are $O_1 \overset{1,0}{\prec} O_2$, $O_1 \overset{1,0}{\prec} O_3$, $O_2 \overset{3,0}{\prec} O_3$, $O_3 \overset{1,2}{\prec} O_1$.

2.2 Resource-Constrained Modulo Scheduling Problem Statement

A *modulo scheduling problem* is a cyclic scheduling problem where all operations have the same processing period $\lambda \in \mathbb{N}$, also called the *initiation interval*. Compared to cyclic scheduling problems, a main simplification is that modulo scheduling problems only need to consider the set of generic operations $\{O_i\}_{1 \leq i \leq n}$. Precisely, by introducing the *modulo schedule dates* $\{\sigma_i\}_{1 \leq i \leq n}$ such that $\forall i \in [1, n], \forall k > 0 : \sigma_i^k = \sigma_i + (k-1)\lambda$, the uniform dependence constraints become:

$$O_i \xrightarrow{\theta_i^j, \omega_i^j} O_j \Longrightarrow \sigma_i^k + \theta_i^j \leq \sigma_j^{k+\omega_i^j} \Longrightarrow \sigma_i + \theta_i^j - \lambda\omega_i^j \leq \sigma_j$$

Let $\{\sigma_i\}_{1 \leq i \leq n}$ denote the modulo schedule dates of a set of generic operations $\{O_i\}_{1 \leq i \leq n}$. A *resource-constrained modulo scheduling problem* (RCMSP) is defined by [6]:

- Uniform dependence constraints: for each such dependence $O_i \xrightarrow{\theta_i^j, \omega_i^j} O_j$, a valid modulo schedule satisfies $\sigma_i + \theta_i^j - \lambda\omega_i^j \leq \sigma_j$. The dependence graph without the dependences whose $\omega_i^j > 0$ is a directed acyclic graph (DAG).
- Modulo resource constraints: each operation O_i requires $\vec{b}_i \geq \vec{0}$ resources for all the time intervals $[\sigma_i + k\lambda, \sigma_i + k\lambda + p_i - 1], k \in \mathbb{Z}$ and the total resource use at any time cannot exceed a given availability $\vec{B}$. The integer value p_i is the processing time of O_i.

The primary objective of resource-constrained modulo scheduling problems is to minimize the period λ. The secondary objective is usually to minimize the iteration makespan. In contexts where the number of processor registers is a significant constraint, secondary objectives such as minimizing the cumulative register lifetimes [5] or the maximum register pressure [7] are considered. This is not the case for the ST200 VLIW processors, so we shall focus on makespan minimization.

2.3 Solving Resource-Constrained Modulo Scheduling Problems

Most modulo scheduling problems cannot be solved by classic machine scheduling techniques, as the modulo resource constraints introduce operation resource requirements of unbounded extent. Also, the uniform dependence graph may include circuits (directed cycles), unlike machine scheduling precedence graphs that are acyclic. Even without the circuits, some modulo dependence latencies $\theta_i^j - \lambda\omega_i^j$ may also be negative. Thus, the resource-constrained modulo scheduling problem appears as a Resource-Constrained Project Scheduling Problem with maximum time-lags (RCPSP/max) and modulo resource requirements.

In practice, building modulo schedules with heuristics is much easier than building RCPSP/max schedules. This is because the maximum time-lags of RCMSPs always include a term that is a negative factor of the period λ. Such constraints can always be made redundant by increasing enough the period λ. A similar observation holds for the modulo resource constraints.

In the classic modulo scheduling framework [14, 11, 16], a dichotomy search for the minimum λ that yields a feasible modulo schedule is performed, starting from $\lambda_{\min} \stackrel{\text{def}}{=} \max(\lambda_{\text{rec}}, \lambda_{\text{res}})$ with:

$$\lambda_{\text{rec}} \stackrel{\text{def}}{=} \max_C \left\lceil \frac{\sum_C \theta_i^j}{\sum_C \omega_i^j} \right\rceil : \quad C \text{ dependence circuit}$$
$$\lambda_{\text{res}} \stackrel{\text{def}}{=} \max_{1 \leq r \leq R} \left\lceil \frac{\sum_{i=1}^n p_i b_i^r}{B^r} \right\rceil : \quad R = \dim(\vec{B})$$

That is, λ_{rec} is the minimum λ such that there are no positive length circuits in the dependence graph and λ_{res} is the minimum λ such that the renewable resources $\vec{B}$ are not over-subscribed.

3 Time-Indexed Integer Programming Formulations

3.1 The Non-Preemptive Time-Indexed RCPSP Formulation

Due to its ability to describe the RCPSP with general temporal constraints and other extensions, the *non-preemptive time-indexed formulation* by Pritsker, Watters and Wolfe in 1969 [13] provides the basis for many RCPSP solution strategies. In this formulation, T denotes the time horizon and $\{0,1\}$ variables $\{x_i^t\}$ are introduced such that $x_i^t \stackrel{\text{def}}{=} 1$ if $\sigma_i = t$, else $x_i^t \stackrel{\text{def}}{=} 0$. In particular $\sigma_i = \sum_{t=1}^{T-1} t x_i^t$. The following formulation minimizes the makespan C_{max} by minimizing the completion time of a dummy operation O_{n+1} that is dependent on all the other operations:

$$\sum_{t=1}^{T-1} t\, x_{n+1}^t : \qquad \text{minimize} \tag{1}$$

$$\sum_{t=0}^{T-1} x_i^t \quad = \quad 1 \qquad \forall i \in [1, n+1] \tag{2}$$

$$\sum_{s=t}^{T-1} x_i^s + \sum_{s=0}^{t+\theta_i^j-1} x_j^s \quad \leq \quad 1 \qquad \forall t \in [0, T-1], \forall (i,j) \in E_{dep} \tag{3}$$

$$\sum_{i=1}^{n} \sum_{s=t-p_i+1}^{t} x_i^s\, \vec{b}_i \quad \leq \quad \vec{B} \qquad \forall t \in [0, T-1] \tag{4}$$

$$x_i^t \quad \in \quad \{0,1\} \quad \forall i \in [1, n+1], \forall t \in [0, T-1] \tag{5}$$

The equations (2) ensure that any operation is scheduled once. The inequalities (3) describe the dependence constraints, as proposed by Christofides et al. [3]. Given (2), the inequalities (3) for any dependence $(i,j) \in E_{dep}$ are equivalent to: $\sum_{s=t}^{T-1} x_i^s \leq \sum_{s=t+\theta_i^j}^{T-1} x_j^s \; \forall t \in [0, T-1]$ and this implies $\sigma_i \leq \sigma_j - \theta_i^j$. Finally, the inequalities (4) enforce the cumulative resource constraints for $p_i \geq 1$. The extensions of the RCPSP with time-dependent resource availabilities $\vec{B}(t)$ and resource requirements $\vec{b}_i(t)$ are described in this formulation by generalizing (4) into (6):

$$\sum_{i=1}^{n} \sum_{s=0}^{t} x_i^s\, \vec{b}_i(t-s) \quad \leq \quad \vec{B}(t) \quad \forall t \in [0, T-1] \tag{6}$$

3.2 The Classic Modulo Scheduling Integer Programming Formulation

The classic integer programming formulation of the RCMSP is from Eichenberger and Davidson [8]. This formulation is based on a λ-decomposition of the modulo schedule dates, that is, $\forall i \in [0, n+1] : \sigma_i = \lambda \phi_i + \tau_i, 0 \leq \tau_i < \lambda$. Given an operation O_i, ϕ_i is its *column number* and τ_i is its *row number*. The formulation of Eichenberger and Davidson introduces the time-indexed variables $\{y_i^\tau\}_{1 \leq i \leq n}^{0 \leq \tau \leq \lambda - 1}$ for the row numbers, where $y_i^t \stackrel{\text{def}}{=} 1$ if $\tau_i = t$, otherwise $y_i^t \stackrel{\text{def}}{=} 0$. In particular, $\tau_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau \; \forall i \in [1, n]$. The column numbers $\{\phi_i\}_{1 \leq i \leq n}$ are directly used in this formulation:

$$\sum_{\tau=0}^{\lambda-1} \tau y_{n+1}^\tau + \lambda \phi_{n+1} : \qquad \text{minimize} \tag{7}$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1 \quad \forall i \in [1,n] \tag{8}$$

$$\sum_{s=\tau}^{\lambda-1} y_i^s + \sum_{s=0}^{(\tau+\theta_i^j-1) \bmod \lambda} y_j^s + \phi_i - \phi_j \leq \omega_i^j - \lfloor \tfrac{\tau+\theta_i^j-1}{\lambda} \rfloor + 1 \quad \forall \tau \in [0,\lambda-1], \forall (i,j) \in E_{dep} \tag{9}$$

$$\sum_{i=1}^{n} \sum_{r=0}^{p_i-1} y_i^{(\tau-r) \bmod \lambda} \vec{b}_i \leq \vec{B} \quad \forall \tau \in [0,\lambda-1] \tag{10}$$

$$y_i^\tau \in \{0,1\} \quad \forall i \in [1,n], \forall \tau \in [0,\lambda-1] \tag{11}$$

$$\phi_i \in \mathbb{N} \quad \forall i \in [1,n] \tag{12}$$

In this formulation, λ is assumed constant. Like in classic modulo scheduling, it is solved as the inner step of a dichotomy search for the minimum value of λ that allows a feasible modulo schedule.

3.3 A New Time-Indexed Formulation for Modulo Scheduling

We propose a new time-indexed formulation for RCMSP, based on the time-indexed formulation of RCPSP/max of Pritsker et al. [13]. First consider the modulo resource constraints. Each operation O_i requires $\vec{b}_i$ resources for the dates in $[\sigma_i + k\lambda, \sigma_i + k\lambda + p_i - 1], \forall k \in \mathbb{Z}$ (§ 2.2). The resource requirement function $\vec{b}_i(t)$ of O_i is written $\sum_{k\in\mathbb{Z}} (t \in [k\lambda, k\lambda + p_i - 1])\vec{b}_i$, so (6) become:

$$\begin{aligned}
& \sum_{i=1}^{n} \sum_{k\in\mathbb{Z}} \sum_{s=0}^{t} x_i^s (t - s \in [k\lambda, k\lambda + p_i - 1])\vec{b}_i \leq \vec{B} \quad \forall t \in [0, T-1] \\
\Longrightarrow \quad & \sum_{i=1}^{n} \sum_{k\in\mathbb{Z}} \sum_{s=0}^{t} x_i^s (s \in [t + k\lambda - p_i + 1, t + k\lambda])\vec{b}_i \leq \vec{B} \quad \forall t \in [0, T-1] \\
\Longrightarrow \quad & \sum_{i=1}^{n} \sum_{k\in\mathbb{Z}} \sum_{s=t+k\lambda-p_i+1}^{t+k\lambda} x_i^s \vec{b}_i \leq \vec{B} \quad \forall t \in [0, T-1] \\
\Longrightarrow \quad & \sum_{i=1}^{n} \sum_{k=0}^{\lfloor \frac{T-1}{\lambda} \rfloor} \sum_{s=t+k\lambda-p_i+1}^{t+k\lambda} x_i^s \vec{b}_i \leq \vec{B} \quad \forall t \in [0, \lambda-1]
\end{aligned}$$

To complete this new RCMSP formulation, we minimize the schedule date of operation O_{n+1} and we adapt the inequalities (3) to the modulo dependence latencies of $\theta_i^j - \lambda\omega_i^j$. This yields:

$$\sum_{t=1}^{T-1} t\, x_{n+1}^t : \quad \text{minimize} \tag{13}$$

$$\sum_{t=0}^{T-1} x_i^t = 1 \quad \forall i \in [1, n+1] \tag{14}$$

$$\sum_{s=t}^{T-1} x_i^s + \sum_{s=0}^{t+\theta_i^j-\lambda\omega_i^j-1} x_j^s \leq 1 \quad \forall t \in [0, T-1], \forall (i,j) \in E_{dep} \tag{15}$$

$$\sum_{i=1}^{n} \sum_{k=0}^{\lfloor \frac{T-1}{\lambda} \rfloor} \sum_{s=t+k\lambda-p_i+1}^{t+k\lambda} x_i^s\, \vec{b}_i \leq \vec{B} \quad \forall t \in [0, \lambda-1] \tag{16}$$

$$x_i^t \in \{0,1\} \quad \forall i \in [1, n+1], \forall t \in [0, T-1] \tag{17}$$

4 A Large Neighborhood Search Heuristic

4.1 Variables and Constraints in Time-Indexed Formulations

The time-indexed formulations for the RCMSP (and the RCPSP/max) involve variables and constraints in numbers that are directly related to any assumed earliest $\{e_i\}_{1\leq i\leq n}$ and latest $\{l_i\}_{1\leq i\leq n}$ schedule dates. Indeed, the number of variables is $\sum_{i=1}^{n} l_i - e_i + 1$. Most of the constraints are the dependence inequalities (15), and $e \stackrel{\text{def}}{=} |E_{dep}|$ non transitively redundant dependences appear to generate eT inequalities with $T \stackrel{\text{def}}{=} \max_{1\leq i\leq n} l_i + p_i$. However, the first sum of (15) is 0 if $t > l_i$. Likewise, the second sum of (15) is 0 if $t + \theta_i^j - \lambda\omega_i^j - 1 < e_j$. So (15) is actually equivalent to:

$$\sum_{s=t}^{T-1} x_i^s + \sum_{s=0}^{t+\theta_i^j-\lambda\omega_i^j-1} x_j^s \quad \leq \quad 1 \quad \forall t \in [e_j - \theta_i^j + \lambda\omega_i^j + 1, l_i], \forall (i,j) \in E_{dep} \tag{18}$$

So (18) is redundant whenever $e_j - \theta_i^j + \lambda\omega_i^j \geq l_i$ ($e_j - \theta_i^j \geq l_i$ in case of the RCPSP/max).

The time-indexed formulation for the RCMSP (and the RCPSP/max) is therefore likely to become more tractable after reducing the possible schedule date ranges $\sigma_i \in [e_i, l_i]$. The basic technique is to initialize $e_i = r_i$ and $l_i = d_i - p_i$, with $\{r_i\}_{1\leq i\leq n}$ and $\{d_i\}_{1\leq i\leq n}$ being the release dates and the due dates, then propagate the dependence constraints using a label-correcting algorithm. More elaborate techniques have been proposed for the RCPSP [4]. Ultimately, all these margins reduction techniques rely on some effective upper bounding of the due dates $\{d_i\}_{1\leq i\leq n}$.

4.2 A Large Neighborhood Search Heuristic for Modulo Scheduling

In case of the RCMSP, the primary objective is the period minimization. Heuristics build modulo schedules at some period λ that is often greater than the lower bound $\lambda_{\min} \stackrel{\text{def}}{=} \max(\lambda_{\text{rec}}, \lambda_{\text{res}})$. When this happens, the feasibility of modulo scheduling at period $\lambda - 1$ is open. Moreover, it is not known how to bound the makespan at period $\lambda - 1$ given a makespan at period λ. This means that effective upper bounding of the due dates in RCMSP instances is not available either.

To compensate for the lack of upper bounding, we propose a large neighborhood search (LNS) for the RCMSP, based on adaptive margins reduction and implicit enumeration of the resulting time-indexed integer programs (using a MIP solver). The LNS [17] is a meta-heuristic where a large number of solutions in the neighborhood of an incumbent solution are searched by means of branch and bound, constraint programming, or integer programming. The large neighborhood is obtained by fixing some variables of the incumbent solution while releasing others.

In the setting of time-indexed formulations, we consider as the neighborhood of an incumbent solution $\{\sigma_i\}_{1\leq i\leq n}$ some schedule date ranges or *margins* $\{e_i, l_i : \sigma_i \in [e_i, l_i]\}_{1\leq i\leq n}$, which are made consistent under dependence constraint propagation. For each $x_i^{\sigma_i} = 1$, we fix variables $x_i^{r_i} \ldots x_i^{e_i-1}$, $x_i^{l_i+1} \ldots x_i^{d_i-p_i}$ to zero and release variables $x_i^{e_i} \ldots x_i^{l_i}$. The fixed variables and the dependence constraints (18) found redundant given the margins are removed from the integer program.

We adapt the generic LNS algorithm of [12] by using margins to define the neighborhoods and by starting from a heuristic modulo schedule. The period λ is kept constant while this algorithm searches increasingly wider margins under a time budget in order to minimize the makespan. The key change is to replace the diversification operator of [12] by a decrement of the period to $\lambda - 1$ while keeping the schedule dates. This yields a pseudo-solution that may no longer be feasible due to the period change. Computational experience shows that a new solution at period $\lambda - 1$ can often be found in the neighborhood of this pseudo-solution, whenever the problem instance is feasible at period $\lambda - 1$. In case a solution at period $\lambda - 1$ is found, go back to minimize the makespan.

4.3 Experimental Results from the ST200 Production Compiler

We implemented the two time-indexed formulations of RCMSP described in Section 3 in the production compiler for the STMicroelectronics ST200 VLIW processor family [6], along with the proposed LNS heuristic, then used the CPLEX 9.0 Callable Library to solve the resulting MIPs.

The table below reports experimental data for the largest loops that could not be solved with these formulations, assuming a timeout of 300s and a time horizon of $4\lambda_{\min}$. Column *#O,#D* gives the number of operations and of non-redundant dependences. Column *Heuristic* λ, M displays the period and makespan computed by the ST200 production compiler insertion scheduling heuristic [6]. The column groups *Formulation 300*, *Formulation 30*, *Eichenberger 300*, correspond to the proposed LNS using our formulation and Eichenberger and Davidson formulation for timeout values of 300s, 30s, 300s. In each group, column *#V,#C* gives the number or variables and constraints of the integer program sent to CPLEX 9.0. In all these cases, the *Formulation* LNS reached the $\lambda_{\min}$.

		Heuristic	Formulation 300		Formulation 30		Eichenberger 300	
Loop	#O,#D	λ, M	λ, M	#V,#C	λ, M	#V,#C	λ, M	#V,#C
q_plsf_5.0_215	231,313	81,97	75,77	1114,1236	75,78	1873,2042	*,*	18942,25989
q_plsf_5.0_227	121,163	42,92	39,46	982,1228	39,46	1378,1685	39,47	4840,6673
q_plsf_5.0_201	124,168	42,92	40,47	1086,1340	40,65	1197,1421	41,50	5208,7217
q_plsf_5.2_11	233,317	82,100	75,78	1113,1216	75,79	1897,2045	*,*	19339,26637
subbands.0_196	130,234	44,65	35,49	718,906	35,48	1008,1248	*,*	5850,10778
transfo.IMDCT_L	232,370	71,109	58,58	1133,1075	58,58	1985,1961	70,74	16472,26482

5 Summary and Conclusions

The resource-constrained modulo scheduling problem (RCMSP) is a resource-constrained cyclic scheduling problem whose solutions must be 1-periodic of integral period λ. The primary minimization objective of the RCMSP is the period and the secondary objective is the makespan. Given any period λ, the RCMSP appears as a resource-constrained project scheduling project with maximum times lags (RCPSP/max) and so-called modulo resource constraints.

Based on the RCPSP/max integer programming formulation of Pritsker et al. [13] and the strong dependence equations of Christofides et al. [3], we present a new time-indexed integer programming formulation for the RCMSP. This formulation differs from the classic time-indexed integer programming formulation of the RCMSP by Eichenberger and Davidson [8].

Both formulations of the RCMSP are impractical to solve for problems that comprise over several tenths of operations, so we propose a large neighborhood search (LNS) heuristic based on solving those integer programming formulations by implicit enumeration after adapting the operation margins. Experiments show this LNS heuristic is quite effective to find a solution at period $\lambda - 1$ given an incumbent solution at period λ, even for RCMSP instances that comprise hundreds of operations. To our knowledge, this is the first application of LNS to cyclic scheduling.

References

[1] V.H. Allan, R.B. Jones, R.M. Lee, S.J. Allan (1995), Software Pipelining, *ACM Computing Surveys* **27**(3), 367 – 432.

[2] P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch (1999), Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods, *European Journal of Operational Research* **112**.

[3] N. Christofides, R. Alvarez-Valdés, J. M. Tamarit (1987), Project Scheduling with Resource Constraints: A Branch and Bound Approach, *European Journal of Operational Research* **29**.

[4] S. Demassey, C. Artigues, P. Michelon (2005), Constraint-Propagation-Based Cutting-Planes: An Application to the Resource-Constrained Project Scheduling Problem, *INFORMS Journal on Computing* **17**(1).

[5] B. Dupont de Dinechin (1994), An Introduction to Simplex Scheduling, *1994 International Conference on Parallel Architecture and Compiler Techniques – PACT'94*.

[6] B. Dupont de Dinechin (2004), From Machine Scheduling to VLIW Instruction Scheduling, *ST Journal of Research* **1**(2), `http://www.st.com/stonline/press/magazine/stjournal/vol0102/`.

[7] A. E. Eichenberger, E. S. Davidson, S. G. Abraham (1995), Optimum Modulo Schedules for Minimum Register Requirements, *International Conference on Supercomputing – ICS*.

[8] A. E. Eichenberger, E. S. Davidson (1997), Efficient Formulation for Optimal Modulo Schedulers, *SIGPLAN Conference on Programming Language Design and Implementation – PLDI'97*.

[9] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, F. Homewood (2000), Lx: a Technology Platform for Customizable VLIW Embedded Processing, *27th Annual International Symposium on Computer Architecture – ISCA'00*.

[10] C. Hanen, A. Munier (1995), A Study of the Cyclic Scheduling Problem on Parallel Processors, *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science* **57**.

[11] M. Lam (1988), Software Pipelining: an Effective Scheduling Technique for VLIW Machines, *SIGPLAN Conference on Programming Language design and Implementation – PLDI'88*.

[12] M. Palpant, C. Artigues, P. Michelon (2004), LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighborhood Search, *Annals of Operations Research* **131**.

[13] A. A. B. Pritsker, L. J. Watters, P. M. Wolfe (1969), Multi-Project Scheduling with Limited Resources: A Zero-One Programming Approach, *Management Science* **16**.

[14] B. R. Rau, C. D. Glaeser (1981), Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing, *14th Annual Workshop on Microprogramming – MICRO-14*.

[15] B. R. Rau, J. A. Fisher (1993), Instruction-Level Parallel Processing: History, Overview, and Perspective, *Journal of Supercomputing* **7**(1-2), 9 – 50.

[16] B. R. Rau (1996), Iterative Modulo Scheduling, *The International Journal of Parallel Processing* **24**(1).

[17] P. Shaw (1988), Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, *Proc. of 4th International Conference Principles and Practice of Constraint Programming - CP98*.

Scheduling and Location (ScheLoc): Makespan Problem with Variable Release Dates

Donatas Elvikis, Horst W. Hamacher, Marcel T. Kalsch
Department of Mathematics, University of Kaiserslautern, Kaiserslautern, Germany,
{elvikis, hamacher, kalsch}@mathematik.uni-kl.de

While in classical scheduling theory the locations of machines are assumed to be fixed we will show how to tackle location and scheduling problems simultaneously. Obviously, this integrated approach enhances the modeling power of scheduling for various real-life problems. In this paper, we present in an exemplary way theory and a solution algorithm for a specific type of a scheduling and a rather general, planar location problem, respectively. More general results and a report on numerical tests will be presented in a subsequent paper.

Keywords: Machine Scheduling, Location Theory, Algorithmics, Gauge Distances

1 Introduction

Scheduling and location theory are equally important areas of operations research with a wealth of applications. For many of these applications it is obvious, that dealing with these problems in the usual *sequential* manner (i.e., taking the output of one of the problems as input of the other) weakens the model and should be replaced by an *integrated* approach (i.e., solving both problems simultaneously). The latter problem, which we call *ScheLoc* was first introduced by Hennes and Hamacher [4] where machines can be located anywhere on a network. A more detailed investigation on this type of ScheLoc was given by Hennes [3].

The focus of this study is to investigate and analyze planar ScheLoc (P-ScheLoc) problems, where machines can be located anywhere in a given planar region. In this short note, we restrict ourselves to the single machine case. Starting from a general formulation and the main concepts of this new class, a specific P-ScheLoc problem – makespan problem with variable release dates – is discussed in detail. We give a first formal description of this problem and derive two conditions to detect optimal solutions directly. Moreover, we present a problem reformulation using a modified version of the Earliest Release Date (ERD) rule. An important tool for solving this problem is the construction of release date bisectors and ordered regions. In our problem formulation the release dates are shown to be be representable by a special type of distance functions, so-called gauges. Using these results, we develop an efficient solution algorithm based on Linear Programming (LP) for polyhedral gauges, which also include as special cases the rectilinear and maximum distances. Finally, complexity results and some concluding remarks are presented.

The results are based on diploma theses of Elvikis [1] and Kalsch [5].

2 Basics

We are given a set $\mathcal{J} = \{1, \ldots, n\}$ of *jobs* with nonnegative *processing times* p_i, $i \in \mathcal{J}$, which must be scheduled nonpreemptively on a single machine M. In addition, we assume that M can be placed anywhere in the plane $\mathbb{R}^2$ and that each job $i \in \mathcal{J}$ has a given *storage location* $a_i \in \mathbb{R}^2$. Hence the general *Single Machine Planar ScheLoc (1-P-ScheLoc) Problem* consists of choosing a *machine location* $X \in \mathbb{R}^2$, under the constraint that the set of jobs $\mathcal{J}$ is completely processed and that all processing conditions are satisfied. Our goal is to optimize some scheduling objective function which depends not only on the sequence of jobs, but also on the choice of X.

In 1-P-ScheLoc problems, each job $i \in \mathcal{J}$ is additionally characterized by the following parameters. The *storage arrival time* $\sigma_i \geq 0$ represents the time at which job i is available at its storage location a_i. If $\sigma_i = 0$, then i is already available at its storage at the beginning of the processing sequence. The *travel speed* $\nu_i > 0$ represents the rate of motion of job i , or equivalently the rate of change of position, expressed as distance per unit time. Hence after job i is available at its storage location a_i, we can start to move i from its storage to the machine M. The time at which i can start its processing is given by its arrival time at M. It is obvious that this time can be interpreted as the job release date. Now, let $dist_i$ be a general distance function on $\mathbb{R}^2$ corresponding to a_i and $\tau_i := \frac{1}{\nu_i} > 0$, then $r_i(X) := \sigma_i + \tau_i \, dist_i(a_i, X)$ is called the *variable release date* of job i for M dependent on its machine location $X \in \mathbb{R}^2$. Moreover, the sequence in which the jobs are to be processed on the machine is defined by a permutation π of $\{1, \ldots, n\}$, where $\pi(j) = i$ means that job i is the j^{th} job in the processing order. The set of all permutations of $\{1, \ldots, n\}$ is denoted by Π_n. Then for each sequence $\pi \in \Pi_n$ and each machine location $X \in \mathbb{R}^2$, we can easily calculate the completion times for all jobs $i \in \mathcal{J}$ using the following recursive formula

$$C_{\pi(1)}(X) = r_{\pi(1)}(X) + p_{\pi(1)}, \tag{1}$$

$$C_{\pi(j)}(X) = \max\{C_{\pi(j-1)}(X) \, , \; r_{\pi(j)}(X)\} + p_{\pi(j)} \quad \forall j \in \{2, \ldots, n\}, \tag{2}$$

where $p_{\pi(j)}$ defines the processing time of job $\pi(j)$. Finally, the maximum completion time (or makespan) in $X \in \mathbb{R}^2$ is given by

$$C_{max}(X) = \max\{C_1(X), \ldots, C_n(X)\} = C_{\pi(n)}(X). \tag{3}$$

To illustrate the modeling potential of this approach, we concentrate on a specific 1-P-ScheLoc problem, the makespan problem with variable release dates.

3 The Problem

In general, the single machine makespan problem with variable release dates (*1-MPVRD*) can be formulated using (1)-(3):

$$\begin{aligned}
\min \quad & C_{\pi(n)}(X) \\
\text{s.t.} \quad & C_{\pi(j)}(X) \geq C_{\pi(j-1)}(X) + p_{\pi(j)} \quad \forall j \in \{2, \ldots, n\} && (4)\\
& C_{\pi(j)}(X) \geq r_{\pi(j)}(X) + p_{\pi(j)} \quad \forall j \in \{1, \ldots, n\} && (5)\\
& \pi \in \Pi_n && (6)\\
& X \in \mathbb{R}^2 && (7)
\end{aligned}$$

where completion time formula (1)-(2) is explicitly represented by constraints (4)-(5). It easy to see that if $p_i = 0$ for all $i \in \mathcal{J}$, then 1-MPVRD reduces to a classical 1-center facility location problem. If we fix X a priori, then we only have to solve a classical makespan problem with fixed release dates. Furthermore, for a given sequence $\pi \in \Pi_n$, we only have to solve a 1-facility location problem to obtain an optimal machine location. For convex distance functions and a fixed sequence $\pi \in \Pi_n$, it is obvious that the objective function $C_{\pi(n)}(X)$ is convex on $\mathbb{R}^2$. Note that, (5) can be replaced by $C_i(X) \geq r_i(X) + p_i$ for all $i \in \mathcal{J}$.

The following two sufficient criteria describe situations, where one of the job locations is an optimal ScheLoc location for the machine. They are proved using the trivial lower bound $LB := \min_{i \in \mathcal{J}}\{\sigma_i\} + \sum_{i \in \mathcal{J}} p_i$.

Proposition 1. *If there exists a job $i \in \mathcal{J}$ with $i \in argmin\{\sigma_s : s = 1, \ldots, n\}$ and $\sigma_i + p_i \geq r_s(a_i) = \sigma_s + \tau_s\, dist_s(a_s, a_i)$ for all $s \in \{1, \ldots, n\}$ then a_i is an optimal machine location and $\pi^* = (i, \pi^*(2), \pi^*(3), \ldots, \pi^*(n)) \in \Pi_n$ with $\pi^*(s) \neq i$, $s \in \{2, \ldots, n\}$, defines an optimal job sequence.*

Proposition 2. *Let $\pi^* \in \Pi_n$ be an optimal sequence in $X = a_i$ with $i \in argmin\{\sigma_s : s = 1, \ldots, n\}$. If $\sigma_i + \sum_{j=1,\ldots,l} p_{\pi^*(j)} \geq r_{\pi^*(l+1)}(a_i)$ for all $l \in \{1, \ldots, n-1\}$, then a_i is an optimal machine location.*

In the following, we assume that neither of the preceding conditions hold such that we have to develop an efficient algorithm to solve ScheLoc.

Recall that 1-MPVRD reduces for a given machine location $X \in \mathbb{R}^2$, to a classical makespan problem with fixed release dates $r_i(X) = r_i$, $i \in \mathcal{J}$. In this case, we can use the well-known ERD rule to obtain an optimal job sequence. Thus, for every machine location $X \in \mathbb{R}^2$ we can easily obtain an optimal job sequence $\pi \in \Pi_n$ using the ***ScheLoc ERD rule***: For machine location $X \in \mathbb{R}^2$, sort the jobs $i \in \mathcal{J}$ in increasing order of their release dates $r_i(X)$, i.e., $r_{\pi(1)}(X) \leq \ldots \leq r_{\pi(n)}(X)$. Thus, 1-MPVRD can be reformulated using the provided ScheLoc ERD rule:

$$\begin{aligned} \min \quad & C_{\pi(n)}(X) \\ \text{s.t.} \quad & (4) - (7) \\ & r_{\pi(1)}(X) \leq \ldots \leq r_{\pi(n)}(X) \end{aligned} \tag{8}$$

Here it should be noted that the objective function is in general non-convex on $\mathbb{R}^2$ (see Example 1 and Figure 3).

Example 1. Consider two jobs with storage locations $a_1 = (0, 0)$ and $a_2 = (10, 5)$ with rectilinear distance l_1. Moreover, let $p_1 = 1$ and $p_2 = 15$, $\sigma_1 = \sigma_2 = 0$ and $\nu_1 = \nu_2 = 1$:

$$C_{max}(a_1) = \max\{r_2(a_1), r_1(a_1) + p_1\} + p_2 = \max\{15, 0 + 1\} + 15 = 30$$

$$C_{max}(a_2) = \max\{r_1(a_2), r_2(a_2) + p_2\} + p_1 = \max\{15, 0 + 15\} + 1 = 16$$

$$C_{max}(0.5 \cdot (a_1 + a_2)) = \max\{7.5, 7.5 + 1\} + 15 = \max\{7.5, 7.5 + 15\} + 1 = 23.5 > \frac{30 + 16}{2} = 23$$

If we assume that our distance functions are convex, then it is easy to see that the objective function is also convex in each of the regions in which the sequence of inequalities (8) does not change.

4 Geometrical Properties: Bisectors and Ordered Regions

Let $i, j \in \mathcal{J}$ with $i \neq j$. Then the set $B^{i,j} := \{X \in \mathbb{R}^2 \mid r_i(X) = r_j(X)\}$ is called the *release date bisector* with respect to job i located in a_i and job j located in a_j.

The bisectors divide the plane into various *(release date -) ordered regions* $O_\pi := \{X \in \mathbb{R}^2 \mid r_{\pi(1)}(X) \leq \ldots \leq r_{\pi(n)}(X)\}$ defined by permutations $\pi \in \Pi_n$ (see Figure 1). In each O_π, the order of the release dates does not change. Note that, ordered regions are in general neither convex nor connected.

For each $X \in O_\pi$ an optimal job sequence of problem 1-MPVRD is obtained by π. Thus solution of a location problem for all $n!$ permutations of possible ordered regions solves the ScheLoc problem 1-MPVRD. As we will show subsequently, the efficiency of this approach follows, since for a large class of distance functions, only a ***polynomial number of these ordered regions*** needs to be considered, since for many sequences π we have that $O_\pi = \emptyset$, which means that these π can not be optimal sequences.

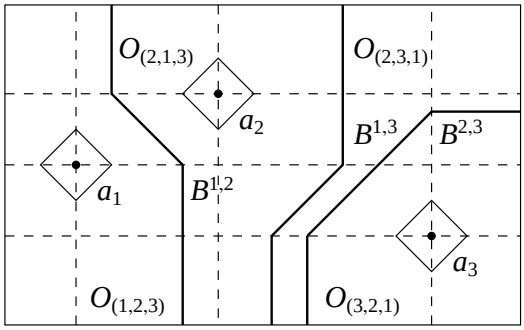


Figure 1: Fundamental directions, bisectors and ordered regions generated by a_1, a_2, a_3 associated with $dist_i = l_1$, $i = 1, 2, 3$. Ordered regions $O_{(3,1,2)}$ and $O_{(1,3,2)}$ are empty.

The considered class of distance functions is the class of polyhedral gauges with respect to a_i, $i \in \mathcal{J} = \{1, \ldots, n\}$, defined by $\gamma_{\mathcal{B}_i}(X) := \inf\{\lambda > 0 \mid X \in \lambda\mathcal{B}_i\}$ (see e.g. Minkowski [6], Nickel and Puerto [7]). Here $\mathcal{B}_i$ is the *unit ball* of the gauge given by a polytope in $\mathbb{R}^2$, i.e., a convex, compact polyhedral set containing the origin in its interior. A polyhedral gauge is a convex distance function and even a norm (called *block norm*), if it is additionally symmetric. Examples for block norms are the rectilinear distance l_1 and the maximum distance l_∞, both having polyhedral unit balls ($\mathcal{B}_{l_1}$ and $\mathcal{B}_{l_\infty}$) with four extreme points.

Denote the set of *extreme points* of the polytope $\mathcal{B}_i \subseteq \mathbb{R}^2$ by $Ext(\mathcal{B}_i) = \{e_g^i \mid g = 1, \ldots, G_i\}$. Moreover, we define $\mathcal{G}_i := \{1, \ldots, G_i\}$, $i \in \{1, \ldots, n\}$, and $G := \max\{G_i \mid i = 1, \ldots, n\}$. The half-lines ξ_g^i, $g \in \mathcal{G}_i$, $i \in \{1, \ldots, n\}$, starting at the origin 0 and passing through an extreme point $e_g^i \in Ext(\mathcal{B}_i)$ are called *fundamental directions*. Moreover, we define Γ_g^i as the *fundamental cone* generated by two consecutive fundamental directions ξ_g^i and ξ_{g+1}^i, where $\xi_{G_i+1}^i := \xi_1^i$. Clearly, $\bigcup_{g \in \mathcal{G}_i} \Gamma_g^i = \mathbb{R}^2$ for every $i \in \{1, \ldots, n\}$ (see Figure 2).

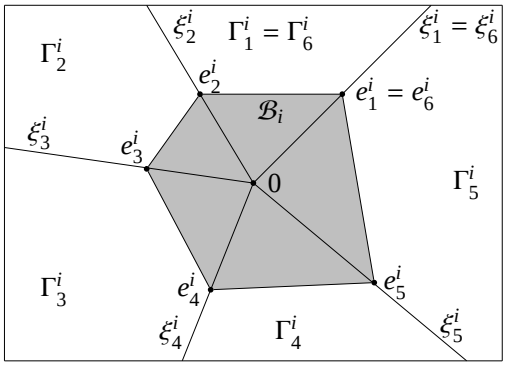


Figure 2: Fundamental directions and cones generated by the extreme points of the convex polyhedron $\mathcal{B}_i$.

The *polar set* $\mathcal{B}_i^\circ$ of $\mathcal{B}_i$ is defined by $\mathcal{B}_i^\circ := \{X \in \mathbb{R}^2 \mid < X, p > \leq 1, \ \forall\, p \in \mathcal{B}_i\}$. (Here and in the following, we use the denotation $< X, p >$ for the inner product $x_1p_1 + x_2p_2$ in $\mathbb{R}^2$.) Its set of extreme points is denoted by $Ext(\mathcal{B}_i^\circ) = \{e_g^{i^\circ} \mid g = 1, \ldots, G_i\}$. For example, the polar set corresponding to $\mathcal{B}_{l_1}$ is $\mathcal{B}_{l_\infty}$, and vice versa.

Lemma 1. (Ward and Wendell [10]) *For all $i \in \{1, \ldots, n\}$ and $X \in \mathbb{R}^2$ the polyhedral gauge $\gamma_{\mathcal{B}_i}(X)$ can be computed by $\gamma_{\mathcal{B}_i}(X) = max\{< e_g^{i^\circ}, X > \mid \ e_g^{i^\circ} \in Ext(\mathcal{B}_i^\circ)\}$.*

Lemma 2. (Thisse et al. [9]) *For $i \in \{1, \ldots, n\}$ let $\mathcal{B}_i \subseteq \mathbb{R}^2$ be a polytope and $\gamma_{\mathcal{B}_i}$ its corresponding polyhedral gauge. Then $\gamma_{\mathcal{B}_i}$ is a linear function on every fundamental cone Γ_g^i, $g \in \mathcal{G}_i$.*

The ***region-wise linearity of polyhedral gauges*** is one of the reasons why the ScheLoc algorithm of this paper is efficient. The other is the fact that only polynomially many regions (sequences) need to be considered in our 1-MPVRD ScheLoc problem.

Theorem 1. *The number of nonempty ordered regions* $|\ \Pi_n^{ord} := \{\pi \in \Pi_n \mid O_\pi \neq \emptyset\}\ |$ *is polynomially bounded by* $O(n^4G^2)$.

Rodríguez-Chía et al. [8] proved this result for polyhedral gauges with multiplicative weights. Since the main argument in their proof is the linearity of polyhedral gauges on every fundamental cone established in Lemma 2, it can easily be extended to release dates, which are generated by polyhedral gauges $\gamma_{\mathcal{B}_i}$ using multiplicative (τ_i) and additional additive (σ_i) weights.

In order to generate the set of sequences Π_n^{ord} we first have to find bisectors for all job pairs $i, j \in \mathcal{J}$ with $i \neq j$. Each bisector can be found in polynomial time (see Weißler [11]). Obviously, bisectors determine the edges of a planar subdivision, whose cells coincide with the nonempty ordered regions. Hence, for all cells O we have to determine the corresponding order π by evaluating and sorting the release dates for some $X \in int(O)$ in increasing order, which leads to set Π_n^{ord}.

The preceding geometrical insights combined with linear programming yield an efficient solution algorithm for ScheLoc. This is shown in the next section.

5 Polynomial ScheLoc Algorithm

For all sequences $\pi \in \Pi_n^{ord}$ consider the following parametric linear program $LP(\pi)$:

$$\begin{aligned}
\min \quad & C_{\pi(n)}(X) \\
\text{s.t.} \quad & C_{\pi(j)}(X) \geq C_{\pi(j-1)}(X) + p_{\pi(j)} \quad \forall j \in \{2, ..., n\} && (9)\\
& C_i(X) \geq \sigma_i + \tau_i \ < e_g^{i^\circ}, X - a_i > + p_i \quad \forall e_g^{i^\circ} \in Ext(\mathcal{B}_i^\circ) \ \ \forall i \in \mathcal{J} && (10)\\
& X \in \mathbb{R}^2 && (11)
\end{aligned}$$

From Lemma 1 we get $r_i(X) = \sigma_i + \tau_i\, \gamma_{\mathcal{B}_i}(X - a_i) = \sigma_i + \tau_i\ \max\{< e_g^{i^\circ}, X > \mid\ e_g^{i^\circ} \in Ext(\mathcal{B}_i^\circ)\}$. For each $\pi \in \Pi_n^{ord}$, let X_π^* be an optimal solution to $LP(\pi)$. If $X_\pi^* \in O_\pi$, then we know that π is a local optimal sequence in X_π^*. If $X_\pi^* \notin O_\pi$, then it is obvious that we can easily find another sequence $\overline{\pi} \in \Pi_n^{ord}$, by evaluating and sorting the release dates in X_π^* in increasing order, such that $C_{\overline{\pi}(n)}(X_\pi^*) \leq C_{\pi(n)}(X_\pi^*)$, which means that π is dominated by $\overline{\pi}$. Thus, for each $\pi \in \Pi_n^{ord}$ we only have to find an optimal machine location by solving the parametric linear program $LP(\pi)$ and output the globally best solution.

The complexity of this algorithm is characterized by the determination of Π_n^{ord} and the complexity of solving the corresponding linear programs $LP(\pi)$. Both can be done in polynomial time.

Example 2. Consider three jobs at storage locations $a_1 = (0, 2)$, $a_2 = (4, 4)$ and $a_3 = (10, 0)$ with rectilinear distance l_1 (see Figure 1 for locations, bisectors and ordered regions). Moreover, let the processing times be $p_1 = 6$, $p_2 = 3$ and $p_3 = 3$, and storage arrival times set to $\sigma_1 = \sigma_2 = 0$ and $\sigma_2 = 3$ with $\nu_1 = \nu_2 = \nu_3 = 1$. Note that the unit ball of l_1 is defined by $\mathcal{B}_{l_1} = \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$ and its corresponding polar set is $\mathcal{B}_{l_1}^o = \mathcal{B}_{l_\infty} = \{(1, 1), (1, -1), (-1, 1), (-1, -1)\}$. By solving parametric LP's for all sequences corresponding to nonempty ordered regions, we get the following objective function values:

$$\pi_1 = (1,2,3): \quad C_3(X^*_{\pi_1}) = 15 \text{ with } X^*_{\pi_1} = (2.3, 1.3)$$
$$\pi_2 = (2,1,3): \quad C_3(X^*_{\pi_2}) = 14 \text{ with } X^*_{\pi_2} = (4.25, 2.25)$$
$$\pi_3 = (2,3,1): \quad C_1(X^*_{\pi_3}) = 17 \text{ with } X^*_{\pi_3} = (6.58, 1.58)$$
$$\pi_4 = (3,2,1): \quad C_1(X^*_{\pi_4}) = 17 \text{ with } X^*_{\pi_4} = (8.7, 0.7)$$

Therefore an optimal job sequence is given by $\pi^* = \pi_2 = (2,1,3)$ with optimal machine location $X^* = (4.25, 2.25)$ and makespan equal to $C_{max}(X^*) = 14$ (see white point in Figure 3). Also note that from the contour surface generated by the objective function, it is obvious that $C_{max}(X)$ is non-convex on $\mathbb{R}^2$.

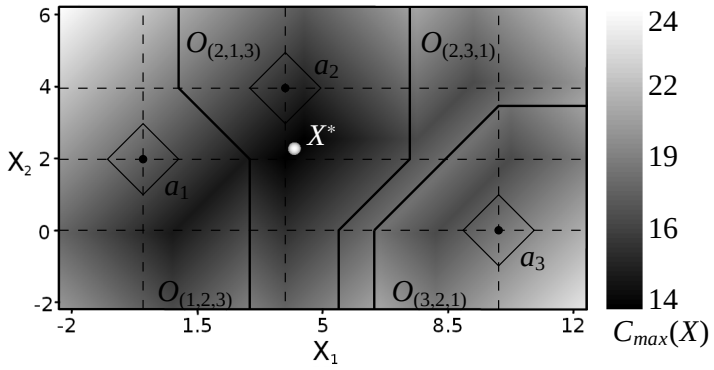


Figure 3: Gauges, ordered regions and optimal location X^* of Example 2. Background is colored with respect to objective function values using scale on the right.

6 Conclusion

The class of ScheLoc problems is a new approach to scheduling with variable machine locations. In this paper we have introduced the 1-P-ScheLoc makespan problem where the release dates are depending on the distance between the (given) locations of the jobs and the (unknown) location of the machine. If this distance is given by polyhedral gauges, we showed that ScheLoc can be reduced to the solution of K linear programs, where K is a polynomial in the number of jobs and extreme points of the unit balls describing the gauge. Special cases include ScheLoc problems with respect to rectilinear or maximum distances.

In [2] we show that the ScheLoc problem introduced in this paper can be considered as a special case of a broader class. In addition to the plane tessellation and LP algorithm we also propose an alternative algorithm which is based on the computation of a finite dominating set (FDS), i.e., a finite set of candidate solutions. Numerical tests will compare the different approaches.

7 Acknowledgement

The authors thankfully acknowledge partial support from the Deutsche Forschungsgemeinschaft (DFG) grant HA 1737/7 "Algorithmik großer und komplexer Netzwerke", New Zealand's Julius von Haast award and the Rheinland-Pfalz cluster of excellence "Dependable adaptive systems and mathematical modeling".

References

[1] D. Elvikis (2006), Implementation of the Linear Programming Approached Algorithm for Solving Single Machine Planar Scheduling Location Makespan Problem, Diploma Thesis, University of Kaiserslautern, Department of Mathematics, Kaiserslautern.

[2] D. Elvikis, H.W. Hamacher and M.T. Kalsch (2007), Simultaneous Scheduling and Location (ScheLoc): Algorithms and Numerical Investigations, Report, University of Kaiserslautern, Department of Mathematics, Kaiserslautern (forthcoming).

[3] H. Hennes (2005), *Integration of Scheduling and Location Models*, Shaker Verlag, Aachen.

[4] H. Hennes and H.W. Hamacher (2002), Integrated Scheduling and Location Models: Single Machine Makespan Problems, Technical Report, University of Kaiserslautern, Department of Mathematics, Report in Wirtschaftsmathematik Nr.82, Kaiserslautern.

[5] M.T. Kalsch (2005), Planar Scheduling Location Problems, Diploma Thesis, University of Kaiserslautern, Department of Mathematics, Kaiserslautern.

[6] H. Minkowski (1967), *Gesammelte Abhandlungen - Band 2.*, Chelsea Publishing Company, New York.

[7] S. Nickel and J. Puerto (2005), *Location Theory: A Unified Approach.*, Springer Verlag, Berlin.

[8] A.M. Rodríguez-Chía, S. Nickel, J. Puerto and F.R. Fernández (2000), A flexible approach to location problems, *Mathematical Methods of Operations Research* **51**(1), 69 – 89.

[9] J.-F. Thisse, J.E. Ward and R.E. Wendell (1984), Some properties of location problems with block and round norms, *Operations Research* **32**, 1309 – 1327.

[10] J.E. Ward and R.E. Wendell (1985), Using block norms for location modeling, *Operations Research* **33**, 1074 – 1090.

[11] A. Weißler (1999), *General bisectors and their application in planar location theory*, Shaker Verlag, Aachen.

Memetic Algorithms and Hyperhill-climbers

Ersan Ersoy
İstanbul Technical University, Informatics Institute, İstanbul, Turkey, eersoy@be.itu.edu.tr
Ender Özcan
Yeditepe University, Computer Engineering Department, İstanbul, Turkey, eozcan@cse.yeditepe.edu.tr
A. Şima Uyar
İstanbul Technical University, Computer Engineering Department, İstanbul, Turkey, etaner@itu.edu.tr

Memetic algorithms (MAs) are meta-heuristics that join genetic algorithms with hill climbing. MAs have recognized success in solving difficult search and optimization problems. Hyperheuristics are proposed as an alternative to meta-heuristics. A hyperheuristic is a mechanism that chooses a heuristic from a set of heuristics, applies it to a candidate solution, and then makes a decision for accepting or rejecting the new solution. In a traditional MA, a single hill climbing method is utilized during the search process. In the presence of multiple hill climbers, the hyperheuristic mechanisms can be adapted for the MAs and employed to exploit the strength of each hill climber better without changing the framework of the MAs. In this study, a set of such mechanisms referred to as *hyperhill-climbers* is investigated for solving exam timetabling problems.

Keywords: Memetic Algorithms, Hill Climbing, Hyperheuristics, Timetabling

1. Introduction

Practical timetabling problems are NP complete constraint optimization problems [16]. A set of events and resources are scheduled subject to a set of constraints. Due to the immense search space and the constraints, traditional approaches might fail in obtaining an optimal solution for a given problem. Numerous approaches, including meta-heuristics and hyperheuristics are proposed for timetabling problems [9]. Ozcan [27] covers different constraint types in timetabling problems. Course timetabling, exam timetabling, sports timetabling and nurse rostering are the most commonly studied problem types in the literature. This study focuses on exam timetabling problems.

Heuristics are used to make a move from a given point in a landscape to another point during the search. Heuristics can be classified as *mutational heuristics* and *hill climbers*. A hill climber is a local search technique that aims to produce a candidate solution with an improved quality whenever applied, whereas mutational heuristics do not have such a purpose. Cowling, et al. [13] describe hyperheuristics as methods that are used to select a low level heuristic from a set of heuristics during the search. In a *simple* hyperheuristic, a single candidate solution representing a problem at hand is continuously processed in an iterative cycle until some termination criteria are satisfied. At each step, a heuristic is selected based on some problem independent criteria, such as the fitness change. Then it is applied to the candidate solution producing a new one. This new solution is accepted or rejected based on a strategy within the hyperheuristic.

Burke et al. [6] provide a hyperheuristic framework that allows the use of any type of heuristics together. Ozcan et al. [29] propose different hyperheuristic frameworks for utilizing hill climbers together with mutational heuristics. One of the proposed frameworks, in which a hill climber is invoked after the application of a selected mutational heuristic, yields an improved performance. Bilgin et al. [4] analyze the performance of thirty five simple hyperheuristics on a set of benchmark functions and a set of exam timetabling benchmark problems by pairing up five *heuristic selection* and seven *move acceptance* strategies.

Genetic Algorithms (GAs), proposed by Holland [20], are inspired from the Darwinian theory of evolution and population genetics. Memetic Algorithms (MAs) [25] are hybrid approaches that use local search (hill climbing) within GAs. A *meme* denotes a hill climbing method. There is strong empirical evidence that the use of a meme improves the performance of a GA [28]. MAs are already used in solving many hard problems, including timetabling problems. Burke et al. discusses the MAs for timetabling in [8]. In [1], [26] and [30], a successful MA is described for dif-

ferent type of timetabling problems. Each MA uses a problem tailored, violation directed mechanism, named as VDHC that manages a set of constraint based hill climbers. This study is an extension to these previous studies. A set of MAs is investigated using multiple hill climbers for solving a set of benchmark exam timetabling problem instances. The MAs include four different VDHC instances and a set of hyperheuristics which will be referred to as hyperhill-climbers used for dynamically determining the suitable hill climber. Moreover, the problem formulation in [12] is used for exam timetabling which is a different formulation than the one used in [30].

2. Preliminaries

Memetic Algorithms (MAs) use local search techniques to improve the exploitation capability of GAs [2],[25]. In a traditional GA, candidate solutions are encoded as *chromosomes* which form the *individuals*. Each individual is made up of *genes*, where each gene receives an *allele* from a set of predetermined values. For example, in a binary encoding an individual is a binary string, where {0,1} is the allele set. In the initial *generation*, a *population* of random individuals is generated. A *fitness function* is used to measure the quality of an individual. In an evolutionary cycle, all individuals go through a set of genetic operations, i.e. *selection*, *crossover* and *mutation*. Depending on the fitness of all the individuals in the population, two of them, termed as *mates*, are randomly selected through a mechanism which favors individuals with better fitness values. *Tournament selection* strategy randomly compares the fitness of a *tournament size* number of individuals and selects the one having the best quality as one of the mates. A *crossover* mechanism is applied to the selected mates, generating new candidate solutions called *offspring*. *One point crossover* randomly determines a crossover location and exchanges the parts of the mates to one side of this point forming two new individuals. The traditional *mutation* mechanism changes each allele to another value from the allele set, usually with a probability of *1/chromosome_length* for each gene in each offspring. Finally, the current population is replaced using individuals from the current generation and from the offspring pool. A *weak elitist* strategy for a *trans-generational* MA (or GA) keeps the best two individuals from the current generation and the rest of the population is filled in from the offspring pool. Evolution terminates whenever some criteria are satisfied.

In a traditional MA, a hill climbing method is applied to each offspring following the mutation step. In this manner, a pool of improved offspring is formed. Notice that in the existence of a set of hill climbers, a mechanism can be used to select one from this set during the improvement stage without changing the original MA framework. For example, Krasnogor formalizes a co-evolutionary framework in [23] as multimeme memetic algorithms and describes a self-adaptive mechanism based on a Lamarckian learning approach. Another possibility is making use of hyper-heuristics for deciding which hill climber to apply whenever necessary. In this study, such mechanisms are investigated. Existing hyperheuristics are adapted to select the best hill climber and the best hill climber orderings within an MA. Cowling et al. discuss most of the simple hyperheuristics in [13]. *Simple Random* (SR) heuristic selection mechanism randomly chooses one of the low level heuristics. *Random Permutation Gradient* (RPG) generates a random permutation of the low level heuristics at first. Then, RPG applies the low level heuristics in turn repeatedly without changing the order of heuristics as long as an improved result is produced. The *Greedy* (GR) method allows all heuristics to process a given candidate solution and chooses the one that generates the most improved solution. *Choice Function* (CF) [22] makes a selection based on the performance history of each low level heuristic and the successively applied pair of low level heuristics. The performance is evaluated using a problem independent measure such as the degree of the previous improvement and the execution time. Gaw et al. [19] study the CF based hyper-heuristics in a different context. Bilgin et al. [4] and Ayob et al. [3] utilize *Improving and Equal* (IE) move acceptance mechanism which rejects only worsening moves. Kendall et al. [21] test the *Great Deluge* (GD) acceptance criterion combined with the SR heuristic selection method as a hyper-heuristic on a set of channel assignment problems. In GD, all moves generating a better or equal objective value than a level computed at each step during the search are accepted. The initial level is set to the objective value of the initial candidate solution. At each step, the level is updated at a linear rate towards the expected objective value.

Except the CF, all strategies can be directly used as a hyperhill-climber within an MA to select the best hill climber whenever required. Each individual is allowed to carry a performance measure along with the genetic material as in multimeme memetic algorithms [23]. Updates are performed in the same way as in [22], [14]. Still, a mechanism is needed for transmitting the measure values of the mates to the offspring. In this study, the simplest strategy is chosen and a randomly selected measure is transmitted to each one. GD requires an initial level to be defined. Traditionally, this level is the fitness of the initial candidate solution in a simple hyperheuristic framework. But, MAs are population based techniques, hence the average fitness of the initial population is used as the initial level in the GD.

3. Memetic Algorithms for Exam Timetabling

Exam timetabling problems (ETPs) require a search for an optimal arrangement of resources for a set of exams based on a set of constraints. The constraints can be divided into two groups; *hard* and *soft* constraints. In a *feasible* solution, all hard constraint must be satisfied. For instance, a student cannot attend two different exams at the same time. The soft constraints are preferences that increase the quality of a solution. For example, increasing the free time between consecutive exams of a student can be considered as a soft constraint. More on exam timetabling can be found in [5], [9], [12], [32]. In this study, the formulation of the exam timetabling problem provided by Carter et al. [12] is used. An optimal schedule for a set of exams is searched for balancing the load of the students subject to the constraint that a student must attend no more than one exam at any time slot (period). An assignment in an ETP is an ordered pair (x,y), where $x \in A$ (set of exams), $y \in B$ (set of periods). The interpretation of this assignment in terms of ETP is: "Exam x starts at time y". Given E exams and a timetable of P periods, the search space size is E^P, hence non-traditional approaches are preferred for solving exam timetabling. The cost function for evaluating the quality of a given solution x is formulated as in Equation 1.

$$f(x) = \frac{1}{S}\sum_{i=1}^{E-1}\sum_{j=i+1}^{E} c_{ij} w_t \qquad where \qquad w_t = \begin{cases} S * 10^6 & t = 0 \\ 2^{5-t} & t = 1,2,3,4,5 \\ 0 & t > 5 \end{cases} \tag{1}$$

E is the number of exams, S is the number of students, c_{ij} is the number of students taking the exams i and j; $i, j \in (1, \ldots, E)$, w_t is the violation weight for t free time slots between exams i and j.

In this study, a traditional MA framework is used as described in Section 2. An allele value indicates the period assigned for the corresponding exam. The chromosome length is equal to the number of exams for a given problem instance. Two different initial population generation methods are implemented. The first method, named as *Largest Degree First* (LDF), schedules the exams in descending order of the number of conflicts. The exam that causes the largest number of conflicts with the other exams is scheduled first. During this process, one of the *available* periods is randomly assigned to the exam. An available period is the one that does not cause an overlap with the previously scheduled exams. If there is no such period, then the exam is randomly scheduled. The second method referred to as *Largest Weighted Degree First* (LWD), schedules the exams in a similar manner. However, instead of using the raw number of conflicts, each exam is weighted by the total number of students involved in the conflicts for that exam. The assignment process is the same as LDF. Both approaches are explained further in [9].

Considering the decomposable cost function in Equation 1, three constraint types can be identified: C1, C2 and C3. C1 is a hard constraint, covering the violations due to conflicting exams which are not supposed to overlap. C2 and C3 represent soft constraints. C2 imposes a condition such that at least three free periods should be between the exams a student should take. Similarly, C3 imposes a condition such that at least six free periods should be between the exams a student should enter. Based on this perspective, three simple hill climbers are implemented to be used within the MA: HC1, HC2 and HC3. Each hill climber attempts to fix the violations due to the corresponding constraint type. Hence, a gradual improvement of individuals is emphasized.

As a data structure, a list for each constraint type is maintained to keep track of the exam pairs which generate a related violation in each individual. A hill climber goes through the relevant list

of all exam pairs one at a time. An exam pair from this list is chosen randomly. Then the randomly selected exam is rescheduled to one of the available periods. If there is no such period, then the other exam goes under the same operation. The lists are updated. All hill climbers operate in the same way. The MA requires a mechanism to select the best hill climber after the mutation step. All mechanisms are organized in three groups. Since, there is a small number of hill climbers, all of them can be applied to an individual in some predefined order successively. The first group of mechanisms contains six permutations of three hill climbers for evaluating the significance of the heuristic orderings. The MAs using such a mechanism is denoted by MA_123, MA_132, MA_213, MA_231, MA_312, and MA_321 where the order of the numbers {1,2,3} indicates the application order of the related hill climber, e.g., MA_123 applies HC1, HC2 and HC3 consecutively.

The second group of hyperhill-climbers arranges the order of the hill climbers based on the number of violations due to each constraint type. The *violation ordering hyperhill-climber* (VIOO) uses the raw number of violations for ordering the hill climbers. The violations are counted for each constraint type and they are sorted from the one that causes the highest number of violations towards the one that causes a lower number of violations. Then the corresponding hill climbers are applied in that order. The *cost ordering hyperhill-climber* (CSTO) uses a similar idea. The weighted cost for each constraint type is utilized instead of the raw violations for sorting the hill climbers. The MAs using these mechanisms are referred to as MA_VIOO and MA_CSTO. Their performances are compared to a MA using the RPG hyperhill-climber (MA_RPG) that determines a random order for HC1, HC2 and HC3 and applies it to an individual. Ozcan describes heuristic templates for timetabling in [27] and [28]. Ozcan et al. use an instance of a heuristic template in [30]. A modified version, denoted as VIOD and CSTD are utilized in this study. Note that this formulation of the exam timetabling problem is different than the previous formulation used in [30]. VIOD and CSTD are violation directed heuristics that organize multiple hill climbers where each one improves a corresponding constraint type in a given problem based on the heuristic template as presented in Figure 1. Three iterations, each of which can be considered as a single stage, are performed in the second while loop 2.a. in Figure 1. In the first stage, the area of concern is marked as all exams. In the second stage, a subset of exams is randomly chosen from the individual. In the last stage, the area of concern is lowered further to a random pair of exams. At each stage a selected hill climber is applied through evaluating the violations due to each constraint type. The probability of a hill climber to be invoked is related to the number of corresponding violations to be satisfied. VIOD uses the raw number of violations during the evaluation for selecting a hill climber, while CSTD uses the weighted cost for that purpose. MA_VIOD and MA_CSTD utilize these heuristics in an MA framework as the third group of hyperhill-climbers.

The last group of mechanisms consists of the hyperhill-climbers as described in Section 2. Two heuristic selection methods (CF, SR) and two acceptance criteria (IE, GD) are paired up. The MAs utilizing the hyperhill-climbers are labeled as MA_CF_IE, MA_SR_IE, MA_SR_GD and MA_CF_GD. The same settings in [22], [14] for CF are used. For a given problem, there might be many hill climbers in which case GR becomes impractical. A modified version, referred to as mGR is used as a hyperhill-climber. In the MA_mGR, two memes are randomly chosen during the hill climbing stage. They are executed separately and the most improved individual is accepted.

In [17], parameter control techniques are classified based on: what is changed (operator probabilities, hill climbing method etc.), how the change is made (deterministic, adaptive, self-adaptive), the scope of the change (population level, individual level, etc.) and the evidence upon which change occurs (monitoring performance of operators, population diversity, etc.). In the deterministic method of changing the parameters, there is a deterministic rule which is used to modify the parameters without using any feedback from the search. In the adaptive mechanisms, the feedback taken from the ongoing search guides the change in the parameters. In self-adaptation, the parameters are coded into the chromosomes and are allowed to evolve along with the individuals. The hyperhill-climbers used within the MAs during this study can be classified based on this terminology. The first group of hyperhill-climbers is deterministic, while the second group of mechanisms is adaptive mechanisms, except the deterministic one RPG. The other hyperhill-climbers in this group adaptively select an appropriate hill climber by dividing a candidate solution into three subparts based on a decomposable penalty oriented fitness function. Hence, a component

level adaptation is employed. In the third group, SR_GD is an adaptive mechanism, operating at individual level, while CF_IE and CF_GD are self-adaptive mechanisms, operating at population level. On the other hand, mGR and SR_IE are deterministic hyperhill-climbers.

1. mark the area of concern as all events
2. **while** (some termination criteria are not satisfied) **do**
    a. **while** (there is improvement and some termination criteria$_2$ are not satisfied) **do**
        i. select a constraint type after evaluating each constraint type violations for the marked events
        ii. apply hill climbing for the selected constraint type to all events within the area of concern
    b. **end while**
    c. lower the area of concern and mark the related events
3. **end while**

Figure 1. Pseudo-code of a heuristic template for timetabling

4. Experimental Results

Six problems from Carter's benchmark [12] are used. The characteristics of each problem instance are presented in Table 1. In the MAs, the population size is fixed as 200, crossover is always applied to selected mates. For each experiment 20 runs are performed**.** A run is terminated whenever the maximum number of generations is exceeded, namely 10,000. All the results are validated via the tool provided at http://www.cs.nott.ac.uk/~rxq/data.htm. As an evaluation criterion, the best fitness value achieved during the runs is used. Additionally, a ranking considering the ties is performed among the compared approaches based on this offline performance criterion. For each experiment, the performance of the different methods will be given on a plot where the x-axis denotes the different methods and the y-axis shows the average ranks over the benchmarks.

Table 1. The characteristics of Carter's benchmarks used during the experiments

Test Case	*No.of Exams*	*No. of Students*	*Enrollments*	*Density of Conflict*	*No. of Periods*
Hec-s-92	81	2823	10632	42.0%	18
Kfu-s-93	461	5349	25113	5.6%	20
Lse-f-91	381	2726	10918	6.3%	18
Sta-f-83	139	611	5751	14.4%	13
Ute-s-92	184	2750	11793	8.5%	10
Yor-f-83	181	941	6034	28.9%	21

During the first part of the experiments, the initialization methods and the predefined hill climber orderings are tested. Each MA using a different hill climber ordering is compared with both initialization methods. The performance of twelve MAs is ranked from 1 to 12, from the best towards the worst for each benchmark data. The MAs using LWD for initialization are illustrated as textured bars. MA_213 using LWD as the initial population generation scheme yields the best performance. It is also observed that using HC3 as the first hill climber generates poor results. LWD is used for initial population generation during the further experiments, since it has a slightly better performance than LDF.

In the second set of experiments, MA_RPG and other mechanisms that rely on the violations due to each constraint type for managing the hill climbers are tested. Experiments are divided into two subparts. In the first part of the experiments, the order of the hill climbers to be applied changes in time. As a hill climber ordering strategy, MA_CSTO provides the best performance on average as illustrated in Figure 3(a). MA_VIOO does not seem to be a viable strategy, since it has a slightly poorer performance than MA_RPG. Narrowing down the area of concern as the hill climbers are applied seems to be an effective strategy. MA_VIOD delivers a better performance as compared to MA_CSTD as shown in Figure 3(a).

The results obtained from the last set of experiments are summarized in Figure 3(b). MA_SR_IE and MA_CF_GD provide the best performance. MA_mGR has the worst performance, although it visits more states as compared to the rest of the MAs in this group of experiments. The approach seems to get stuck at a local optimum due to the intensive use of hill climbers in MA_mGR.

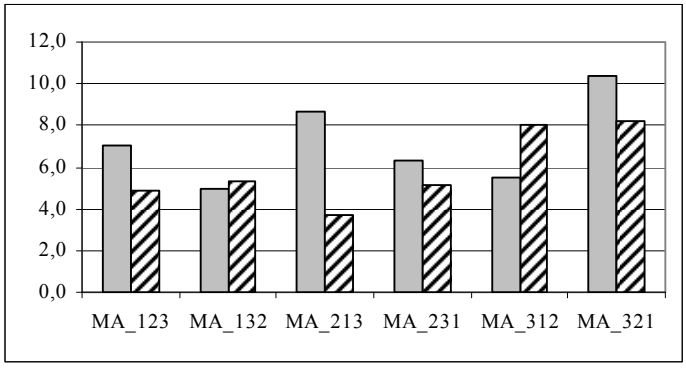
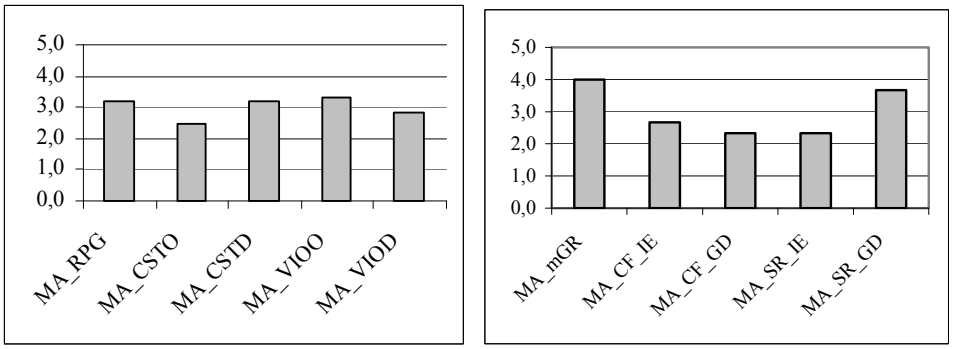


Figure 2. The average rank of each MA using a different hill climber ordering, where the textured bars indicate the MA using LWD and the other using LDF for initialization



(a) (b)

Figure 3. The average rank of each MA with a different hyperhill-climber for (a) the second and (b) third group of experiments.

The best MAs from the experiments and some previous approaches from the literature are compared. The approaches are divided into three groups: deterministic heuristics, stochastic methods, and the best MAs obtained during this study. Due to the parameter variances among the stochastic approaches and different termination criteria, the comparison between the algorithms can be considered as an indirect comparison as presented in Table 2. The hyperhill-climbers within the MAs provide a promising performance. MA_SR_IE turns out to be the best among the MAs, even though due to the termination criteria the maximum number of states it visits is fewer as compared to MA_213, MA_CSTO and MA_VIOD. MA_SR_IE and MA_CF_GD perform better than the LD, SD, LWD and LE heuristics.

Table 2. The performance comparison of the MAs with previously used approaches based on the best fitness values. The performance of seventeen approaches is ranked from 1 to 17, from the best towards the worst for each data and averages are given in the last column (Avr. ranks).

Approaches, [Source]	*Hec-s-92*	*Kfu-s-93*	*Lse-f-91*	*Sta-f-83*	*Ute-s-92*	*Yor-f-83*	*Avr. ranks*
LD, [12]	10.8	14.0	12.0	162.9	38.3	49.9	10.3
SD, [12]	12.7	15.9	12.9	165.7	31.5	44.8	12.6
LWD, [12]	15.8	22.1	13.1	161.5	26.7	41.7	12.5

LE, [12]	15.9	20.8	10.5	161.5	25.8	45.1	11.6
Wal, [33]	12.9	17.1	14.7	158.0	29.0	42.3	13.2
GS, [15]	12.4	18.0	15.5	161.0	29.9	41.0	13.5
Cal, [10]	9.2	13.8	9.6	158.2	24.4	36.2	2.6
BN, [7]	11.3	13.7	10.6	168.3	25.5	36.8	5.9
Mal, [24]	10.6	13.5	10.5	157.3	25.1	37.4	2.2
PS, [31]	10.8	16.5	13.2	158.1	27.8	38.9	8.6
CT, [11]	10.8	14.1	14.7	134.9	25.4	37.5	5.4
MMAS, [18]	11.3	15.0	12.1	157.2	27.7	39.6	6.4
MA_213	11.7	16.0	14.0	157.8	26.3	41.8	9.5
MA_CSTO	11.7	16.1	13.5	158.3	27.2	41.3	10.8
MA_VIOD	11.6	16.5	13.2	158.4	26.7	41.5	10.3
MA_CF_GD	11.8	16.1	13.4	157.7	26.3	40.9	8.8
MA_SR_IE	11.7	15.8	13.3	157.9	26.7	40.7	8.1

5. Conclusions

Hyperheuristics are becoming a central research area beside metaheuristics in search and optimization. This study shows that they can be used as a support mechanism for managing multiple hill climbers within metaheuristics. Memetic algorithms (MAs) as metaheuristics are successfully utilized for solving many difficult problems. In the case of multiple hill climbers, hyperheuristics can be embedded into the MAs as hyperhill-climbers for selecting the best hill climber to apply or deciding the best ordering for successive application of hill climbers. In this study, a set of deterministic, adaptive and self-adaptive hyperhill-climbers are investigated on a benchmark of exam timetabling problem instances, each requiring a satisfactory schedule subject to some constraints. Three hill climbers, each aiming to reduce the violations due to a specific constraint type are utilized. The self-adaptive hyperhill-climbers perform better as compared to the adaptive ones. The CF_GD as a self-adaptive mechanism delivers a good performance. Ordering the hill climbers with respect to the weighted violations for consecutive application seems to be a better choice than ordering them with respect to the raw number of violations. Yet, a deterministic hyperhill-climber, namely, SR_IE that selects a single hill climber at a time turns out to be the best one among all. The experimental results show that the hyperhill-climbers are viable strategies to manage a set of low level hill climbers within an MA. The effect of the number of hill climbers that a hyperhill-climber manages will be investigated further as a future work.

References

[1] A. Alkan, E. Ozcan (2003), Memetic Algorithms for Timetabling. Proc. of IEEE Congress on Evolutionary Computation, 1796 – 1802.

[2] S. Areibi, M. Moussa, H. Abdullah (2001), A Comparison of Genetic/Memetic Algorithms and Other Heuristic Search Techniques, Int. Conf. on Artificial Intelligence, 660 – 666.

[3] M. Ayob, G. Kendall (2003), A Monte Carlo Hyper-Heuristic to Optimise Component Placement Sequencing for Multi Head Placement Machine, Proc. of the Int. Conf. on Intelligent Technologies, 132 – 141.

[4] B. Bilgin, E. Özcan, E.E. Korkmaz (2006), An Experimental Study on HyperHeuristics and Final Exam Scheduling, Proc. of the 6th Int. Conf. on the PATAT, 123 – 140.

[5] E. Burke, D. Elliman, P. Ford, B. Weare (1996), Examination Timetabling in British Universities- A Survey, *Lecture Notes in Computer Science* **1153**, Springer-Verlag, 76 – 90.

[6] E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg (2003), Hyperheuristics: An Emerging Direction in Modern Search Technology, *Handbook of Metaheuristics. International Series in OR & Management Science*, Kluwer Academic Publishers, Boston Dordrecht London, vol. 57, 457 – 474.

[7] E.K. Burke and J. Newall (2003), Enhancing timetable solutions with local search methods. *Lecture Notes in Computer Science* **2740**, Springer-Verlag, 195 – 206.

[8] E.K. Burke, J.D. Landa Silva (2004), The Design of Memetic Algorithms for Scheduling and Timetbaling Problems. Krasnogor N., Hart W., Smith J. (eds.), *Recent Advances in Memetic Algorithms,* Studies in Fuzziness and Soft Computing, vol. 166, Springer, 289 – 312.
[9] E.K. Burke, S. Petrovic (2002), Recent Research Directions in Automated Timetabling, *European Journal of Operational Research* **140**(2), 266 – 280.
[10] M.P. Caramia, Dell'Olmo and G.F. Italiano (2001), New algorithms for examination timetabling. *Lecture Notes in Computer Science* **982**, Springer-Verlag, 230 – 241.
[11] S. Casey and J. Thompson (2003), Grasping the examination scheduling problem. *Lecture Notes in Computer Science* **2740**, Springer-Verlag, 233 – 244.
[12] M.W. Carter, G. Laporte, and S.T. Lee (1996), Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the Operational Research Society* **47**, 373 – 383.
[13] P. Cowling, G. Kendall, E. Soubeiga (2000), A Hyperheuristic Approach to Scheduling a Sales Summit, *LNCS 2079*, Proc. of the 3th Int. Conf. on the PATAT, 176 – 190.
[14] P. Cowling, G. Kendall, E. Soubeiga (2001), A Parameter-free Hyperheuristic for Scheduling a Sales Summit. *Proc. of the 4th Metaheuristic International Conference*, MIC, 127 – 131.
[15] L. Di Gaspero and A. Schaerf (2001), Tabu search techniques for examination timetabling. *Lecture Notes in Computer Science* **2079**, Springer-Verlag, 104 – 117.
[16] S. Even, A. Itai, A. Shamir (1976), On the Complexity of Timetable and Multicommodity Flow Problems, *SIAM J. Computing* **5**(4), 691 – 703.
[17] A.E. Eiben, R. Hinterding, Z. Michalewicz (1999), Parameter Control in Evolutionary Algorithms, *IEEE Trans. Evolutionary Computation* **3**(2), 124 – 141.
[18] M. Eley (2006), Ant Algorithms for the Exam Timetabling Problem, Proc. of the 6th Int. Conf. on the PATAT, 167 – 180.
[19] A. Gaw, P. Rattadilok, R.S.K. Kwan (2004). Distributed Choice Function Hyperheuristics for Timetabling and Scheduling, Proc. of the 5th Int. Conf. on the PATAT, 495 – 498.
[20] J.H. Holland (1975), *Adaptation in Natural and Artificial Systems*, Univ. Mich. Press
[21] G. Kendall, M. Mohamad (2004), Channel Assignment in Cellular Communication Using a Great Deluge Hyperheuristic, *2004 IEEE International Conference on Network*, 769 – 773.
[22] G. Kendall, P. Cowling, E. Soubeiga (2002), Choice Function and Random HyperHeuristics, SEAL'02, 667 – 671.
[23] N. Krasnogor (2002), *Studies on the Theory and Design Space of Memetic Algorithms*, PhD Thesis, University of the West of England, Bristol, UK.
[24] L.T.G. Merlot, N. Boland, B.D. Hughes and P.J. Stuckey, New benchmarks for examination timetabling. Testproblem Database, http://www.or.ms.unimelb.edu.au/timetabling.html.
[25] P. Moscato, M.G. Norman (1992), A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message–Passing Systems, *Parallel Computing and Transputer Applications*, 177 – 186.
[26] E. Ozcan (2005) Memetic Algorithms for Nurse Rostering, P. Yolum (Eds.): *Lecture Notes in Computer Science* **3733**, Springer-Verlag, The 20th ISCIS, 482 – 492.
[27] E. Ozcan, (2005), Towards an XML based standard for Timetabling Problems, TTML, *Multidisciplinary Scheduling, Theory and Applications*, Springer Verlag, 163 – 185.
[28] E. Ozcan (2006), An Empirical Investigation on Memes, Self-generation and Nurse Rostering, Proc. of the 6th Int. Conf. on the PATAT, 246 – 263.
[29] E. Ozcan, B. Bilgin, E.E. Korkmaz (2006). Hill Climbers and Mutational Heuristic, *Lecture Notes in Computer Science* **4193**, Springer-Verlag, PPSN IX, 202 – 211.
[30] E. Ozcan, E. Ersoy (2005), Final Exam Scheduler-FES, Proc. of 2005 IEEE Congress on Evolutionary Computation, vol. 2, 1356 – 1363.
[31] L. Paquete and T. Stuetzle (2002), Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. Proc. of the 4th PATAT, 413 – 420.
[32] R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, S.Y. Lee (2006), *A Survey of Search Methodologies and Automated Approaches for Examination Timetabling*, Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham.
[33] G.M. White, B.S. Xie and S. Zonjic (2004) Using tabu search with long-term memory and relaxation to create examination timetables. *European Journal of Op. Research* **153**, 80 – 91.

Applying Optimal Control to Jetty Scheduling Problems

Fabio Dias Fagundez
Universidade Federal do Rio de Janeiro, LOA/COPPE, Rio de Janeiro, Brazil, fabio.fagundez@gmail.com
João Lauro Dorneles Facó
Universidade Federal do Rio de Janeiro, IM/DCC, Rio de Janeiro, Brazil, jldfaco@acd.ufrj.br

Crude oil and derivatives jetty scheduling problems are modeled by Optimal Control with nonlinear state equations and the use of flow rates as control variables. This paper shows that the Optimal Control formulation is able to represent scheduling problems with continuous variables only. All variables are submitted to lower and upper bounds. Difficulties in the numerical solution of these models are overcome by using an efficient Nonlinear Programming method. Test cases are discussed.

Keywords: scheduling, optimal control, nonlinear programming.

1. Introduction

Planning and scheduling are activities of major economic importance: efficient planning and scheduling can optimize the usage of resources, lead to reduction on waste, and increase of operational profits. They are able to avoid delays and assure that demands are supplied without degrading the quality of services. This paper focuses on the problem of crude oil and derivatives scheduling in ports (the jetty scheduling problem). This article proposes an original approach based on a continuous nonlinear optimal control model, without binary decision variables. We employ the Generalized Reduced Gradient (GRG) method [1] to solve the problem.

2. The Problem

The jetty scheduling of crude oil and derivatives is the problem to determine: (i) tanker vessel (ship) allocation on jetties; (ii) sequence of tanks to load and unload the ships; (iii) sequence of batches in the pipelines (connecting refineries or petrochemical plants to the port); minimizing an objective cost function restricted by constraints of different natures (economical, physicochemical, operational, or environmental). Leffler [2] states that empirical nonlinear equations are often used to model physical phenomena in the oil industry (e.g. blending). Shah [3] proposes a Mixed Integer Linear Programming (MILP) approach for crude oil scheduling from tankers to crude distillation units (CDU): a refinery problem (called the downstream problem) is solved, defining the sequence of crude oil batches in the pipeline and the scheduling of refinery tanks. In sequence, a harbor problem (upstream problem) is solved, defining the allocation of ships on the jetties and the scheduling of harbor tanks, constrained by the pipeline batches defined by the first problem. Más and Pinto [4] present a MILP model as well, dividing the logistic system in three subsystems: (1) harbor, (2) distribution centers (intermediate storage), and (3) refineries (Figure 1). Firstly, they solve the harbor problem, defining the allocation of ships on the jetties, the sequence of crude oil batches in the pipelines, and the scheduling of harbor tanks, constrained by planned quantities of crude oil that have to be sent from the harbor to the pipelines. In sequence, they solve the distribution center problem, constrained by the pipeline batches defined by the harbor problem and known quantities of crude oil that are planned to be consumed by the refineries. They do not deal with the refinery internal scheduling.

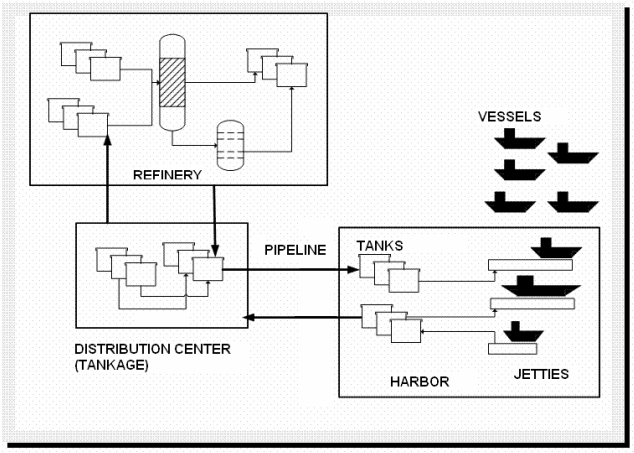


Figure 1 Schematic Representation of the subsystems

The plant or refinery subsystem is very complex and is often divided into internal operational areas, e.g. crude processing, unit operations and blending, storage and delivery of finished products [5, 6]. Distribution centers coordinate the transportation of material among different plants, suppliers and clients, and may be considered as part of the refinery's schedule [3] or not [4]. In this paper we will deal with refineries and distribution centers indirectly, considering that the harbor scheduling must agree to chosen quantities of material through the pipelines.

The harbor has its own tank farm. Therefore, stocking costs have to be considered in the operation. In general, one tank can store only one kind of material, e.g. crude tanks cannot be used for diesel and vice-versa. In the case of finished products, it is common to seal the tank after a laboratorial analysis has assured the product's quality, to avoid any contamination. In the case of crude oil tanks, it is usually necessary to let the oil settle for a time in order to segregate impurities (e.g. brine). The following operational rule is adopted: no tank in the harbor can start a delivery of material if it had not been idle for a certain amount of time. This time is called "settling time" or simply "idle time". Another rule that is commonly adopted is to avoid online blending: do not operate two tanks in parallel, delivering material to a common destination.

Each jetty in the harbor has defined draught and extension. Moreover, pumps may restrict the material that can be pumped. Therefore, ships can berth only on a restricted set of jetties, defined by their cargo and physical dimensions. One ship can berth on a jetty only if the jetty has been free for enough time to allow the previous ship to leave the harbor. Ships are contracted to discharge (or be loaded with) material in a defined time window. Depending on the weather conditions, this window can be shifted but it is considered as known in the schedule. If the vessel is not completely processed until the contractual leaving date, a fee (demurrage) has to be paid.

Along the pipelines, it is important to ensure that consecutive batches of material have similar properties, in order to avoid poor-quality materials contaminating high-quality materials through their interfaces. Operators usually have batch tracking tools to monitor the distribution of the batches inside the pipelines.

3. The Optimal Control Approach

3.1. The General Optimal Control Problem

The General Optimal Control Problem is defined as the problem to minimize a performance objective-function of the state and control trajectories of a given system over the time, subjected to state equations and lower and upper bounds on state and control [7]. States represent the system attributes that we can measure, whereas controls are the actions that we can apply on the system to

change its state. The time can be continuous – leading to a formulation based on differential state equations and the integral of the performance function – or discrete, with intervals of constant or variable sizes Δt_i – leading to a formulation that is equivalent to a Mathematical Programming problem [8] (see Figure 2). If Δt_i is not fixed, it is modeled as one additional control variable per time period.

$$\begin{aligned}
&\textbf{Minimize} && \sum_{t=0}^{T-1} g_t(x_t,u_t) + G_T(x_T),\ g_t \in C^1 \\
&\textbf{S.t.} && x_{t+1} = f_t(x_t,u_t),\ f_t \in C^1 \\
& && x\min_t \le x_t \le x\max_t \\
& && u\min_t \le u_t \le u\max_t \\
& && x_t \in \Re^m, u_t \in \Re^n \\
& && t \in \{0,1,...,T-1\}
\end{aligned}$$

Figure 2 - General formulation for discrete-time Optimal Control problems.

In the discrete-time formulation, the control vector u_t is constant over every time period t, the state vector of period $t+1 (x_{t+1})$ is calculated as a function (f_t) of previous control and state, and the problem's performance function is the summation of performance functions g_t evaluated at all time intervals and G_T at the final state only. State and control variables are bounded by upper and lower bounds, which may be dependent on the given time instant.

3.2. Modeling the Transfer Operation

The fundamental scheduling activity is the transfer operation, i.e., the transportation of a certain quantity of material from one equipment to another during a certain amount of time: a tank or ship is filled by flows from other equipments, changing its state (volume and composition); and this equipment will later perform an outlet transfer operation, performing changes on other equipments.

One way to model this operation is found in [3, 4, 5]: binary variables $b_{ij}(t)$ are mapped to every pair of connected equipments (i,j) for any time period t, and continuous variables $Q_{ij}(t)$ (with lower and upper bounds) to the amount of material that might be transferred from equipment i to j at time t. Constraints like "*At most one equipment can send material to a certain other equipment j at a given instant t* " are modeled as summations of b_{ij} bounded by 1 (one), i.e., pseudo-Boolean representations of propositional logic clauses. The main advantage of this approach is that in MILP models or in MINLP convex models, the optimal solution is, in fact, the global optimal solution. However, real-life instances may become too large, and may not be solved in a reasonable time. Another drawback is that MILP problems may not be able to deal with some empirical equations found in chemical engineering process simulation models, which may be the only way to faithfully represent such processes.

Our proposal is to model this operation in a more compact way: continuous control variables $u_{ij}(t)$ (with lower and upper bounds) are mapped to every pair of connected equipments i,j for any time period t, representing the volume flow rate of material from equipment i to j at time t. Constraints like "*At most one equipment can send material to a certain other equipment j at a given instant t* " are modeled as a summation of bilinear terms $u_{ij}u_{kj}$, $k > i > j$, bounded by 0 (zero) or, in practice, by a tolerance ε. The main advantages of this approach are: (i) real-life problems can be modeled with fewer variables (compared to the previous approach), leading to reasonable computation times; (ii) empirical equations and simulation models can be directly embedded in the model. The main disadvantage is that the use of bilinear terms makes the problem nonconvex. Therefore, there is no guarantee of global optimality of the solution. However, bilinear terms may be linearized in a number of problems [5].

Table 1 compares the size of two models for a Petrobras harbor (GEBAST) in Brazil: the MILP from Más and Pinto in [4], and the Optimal Control model that will be detailed in the next section.

Table 1 - Comparison on GEBAST model sizes

Feature	MILP	Optimal Control
Vessels	13	13
Jetties	4	4
Tanks	18	18
Products	14	14
Product Class	7	7
Pipelines	2	2
Horizon	168h (non-fixed Δt)	168h (Δt = 12h)
0-1	1039	0
Continuous	1996	896
Constraints	7203	5209

3.3. The Optimal Control Model

The identification of the state (x) and the control (u) comes directly from the problem definition: states are consequences of previous states and controls, whereas control is where one can actuate in order to change a future state of the system. The flow rates of transfer operations are modeled as the control, whereas volume and properties are modeled as the state. Berth allocation is also modeled with control variables. For example, if there is a flow from one ship to a tank through a jetty, it means that this ship was allocated to this jetty.

(1) $0 = u_{\min} \le u(t) \le u_{\max}(t)$

Equation (1) shows the control *u(t)* bounded by upper and lower limits. Control bounds (u_{min} and u_{max}) are defined by pumping capacity, maintenance, jetty unavailability because of tides, and forbiddance of ship berthing before its time window.

(2) $x_{\min} = \begin{bmatrix} v_{\min} \\ p_{\min} \end{bmatrix} \le x(t) = \begin{bmatrix} v(t) \\ p(t) \end{bmatrix} \le x_{\max} = \begin{bmatrix} v_{maz} \\ p_{\max} \end{bmatrix}$

Equation (2) shows the state *x(t)* composed by two sets of state variables: *v(t)* and *p(t)*. The first represent the volumes of equipments (tanks, vessels and pipeline), and the second the properties of stored material, such as density, sulphur concentration or product components. There are upper (v_{max}, p_{max}) and lower bounds (v_{min}, p_{min}) to the state, representing storage capacity and product quality specifications.

(3) $v(t+1) = v(t) + \Delta t U u(t)$

Equation (3) is a state equation for volume calculation. As the flow rates are nonnegative, we use the matrix U, whose items are 0, 1 or -1, to update the volumes according to the transfer orientation. It is important to notice that U is defined in a pre-solver phase, based on the graph of connected equipments.

(4) $p_{i,q}(t+1) = f_{i,q}(x_i(t), p_{in,i,q}(t), u_{in,i}(t))$

(5) $f_{i,q}(x_i(t), p_{in,i,q}(t), u_{in,i}(t)) = \dfrac{v_i(t)p_{i,q}(t) + u_{in,i}(t)p_{in,i,q}(t)\Delta t}{v_i(t) + u_{in,i}(t)\Delta t}$

Equation (4) is the state equation for blending calculation and Equation (5) is an example of blending function: $p_{in,i,q}$ is the value of property q being transferred to equipment i at flow rate $u_{in,i}$, while $f_{i,q}$ is a class C^1 blending function valid for volumetric-based properties, such as density and sulfur concentration. Other properties can be converted to linear indexes, and then blended as in Equation (5). One example is the Viscosity Index (VI), as calculated by the norm ASTM D2270 [9]. Other

blending functions may be used for other properties, as long as they show the following characteristic: if $u_{in,i} = 0$, then $p_i(t+1) = p_i(t)$.

Additional constraints can be handled explicitly as new bounded state variables (*z, y, w, r, s*) or implicitly as penalizations in the objective function:

(6) $0 \le z_{j,i}(t+1) = u_{j,i} \sum_{k \neq i} \sum_{t'=t-Tp}^{t} u_{j,k}(t') \le \varepsilon$

Equation (6) models: "*A vessel i can be allocated in a jetty j only if the jetty was empty for the time (Tp) necessary for a previous vessel leave the port*".

(7) $0 \le y_j(t+1) = u_{out,j} \sum_{t'=t-Ts}^{t} u_{in,j}(t') \le \varepsilon$

Equation (7) models: "*A tank j can make a delivery only if it was idle for the necessary settling time (Ts) since its last incoming flow*".

(8) $0 \le w_j(t+1) = \sum_{i \in J} \sum_{k>i \in J} u_i(t) u_k(t) \le \varepsilon$

Equation (8) models: "*At most one equipment can send material to a certain other equipment j at a given instant t*". J is the index set for flows to equipment j.

(9) $0 \le r_n(T) = (v_n(T) - vp_n)^2 \le \varepsilon$

Equation (9) models: "*At the end of the schedule, vessel n must have been properly handled*". The planned final volume of the vessel is vp_n, whereas the actual final volume is $v_n(T)$.

(10) $0 \le s_{n,m}(T) = (v_n(T) - vpp_n)^2$

Equation (10) models: "*At the end of the schedule, the pipeline n must have transported the planned amount vpp_n of material M*".

The objective function is a weighted sum of individual costs (Equation 11), which are calculated over either the final state or the intermediate states.

(11) $F(0,\ldots,T) = \sum_j w_j C_j(x(T),u(T)) + \sum_t \sum_i w_i C_i(x(t),u(t)),\ 0 \le t \le T-1$
(12) $C_1(t) = \sum_n \max(0,t-lt_n) c_n (v_n(t)-vp_n)^2$, lt_n = contracted departure, c_n = unitary cost
(13) $C_2(t) = \sum_n c_n v_n(t)$, c_n = unitary cost
(14) $C_3(t) = \sum_n c_n (u_n(t) - u_n(t-1))^2 (1 + u_n(t-1))^{-1}$, c_n = unitary cost, $t > 0$
(15) $C_4(t) = \sum_n c_n (p_{n,q}(t) - p_{n,q}(t-1))^2 (1 + p_{n,q}(t-1))^{-1}$, c_n = unitary cost, $t > 0$

Equation (12) models the demurrage, Equation (13) inventory costs, Equation (14) penalizes unnecessary changes of flow, and Equation (15) interfaces on consecutive batches in the pipelines.

4. Solving Strategy

Our solving strategy is the following: (i) define the minimum set of control variables from the harbor topology; (ii) initialize the problem with a feasible point (or near feasibility); (iii) relax constraints (6) to (10), penalize them and solve a sequence of relaxed problems, increasing the penalty factor problem by problem. As any non-convex NLP problem, the optimal solution that is found may be a local optimum. Therefore, it is useful to start the solver with different starting points to check if a better local optimum is found.

4.1 Defining control variables

For any given problem, we store the harbor topology in a relational database, with information about tanks, jetties, vessels and pipelines and connections. In a pre-solver phase, a SQL query de-

fines the smallest set of controls: checking pumping capacity against product kind; vessel size, draught, and cargo against jetty size, draft, and tank storage class; etc. The number of free control variables can be further reduced by checking which ones can be fixed at certain time periods.

4.2 GRG Method

Abadie showed in [8] that the GRG method fits well optimal control problems, because they present diagonal-block structured Jacobian matrix of the constraints, and usually many linear constraints. Facó [10] proposed a specialized GRG algorithm for optimal control called GRECO. The GRG converges to a local optimum and divides the variables in two sets: basic and independent variables. The independent variables are manipulated by the method iteratively, whereas the basic variables are calculated from the independent. We employ as GRG solver Smith and Lasdon's Large Scale GRG (LSGRG2) method [X] implemented by Frontline Systems [11]**.**

4.3 Initialization

Because the definition of an initial starting point is very important for nonlinear methods (especially for nonconvex problems), we designed [12] a heuristic procedure able to find feasible points or points with a small number of violated constraints, summarized as the following:

1. For each pipeline: search for a tank to be loaded or unloaded, starting at t = 0;
2. Sort ships by arrival date;
3. For each ship: search for empty jetty and tank at arrival date. If not found, allocate the ship at the end of the scenario.

The step 3 guarantees that all ships will be handled (as long as the scenario does not end), but with the tradeoff of increasing the cost of the objective function. The heuristic does not guarantee that the pipeline plan will be fulfilled.

We proposed in [13] another initialization procedure: start the problem with all control variables set to zero (called a trivial point). It is easy to see that, by relaxing constraints (6) to (10), any point with all control variables set to zero is feasible (but non-optimal).

4.3 Penalty Method

Penalty methods are based on the following function: $F(x) + \mu P(x)$, where $F(x)$ is the objective function, $P(x)$ is the penalty function. We employ the following penalty function, which relaxes constraints (6) to (10):

$$(16) \qquad P(x) = \ln(\|z\|^2+1) + \ln(\|y\|^2+1) + \ln(\|w\|^2+1) + \ln(\|r\|^2+1) + \ln(\|s\|^2+1)$$

We start with $\mu = 0$, solve the relaxed problem, increase μ and solve again, until $P(x) \leq \varepsilon$.

5. Experimental Results

Four different test cases check if our approach can solve problem instances in acceptable time (seconds) in an AMD Athlon XP 2.6GHz, 512MB RAM computer station. Each test case presents a difficulty, or a typical situation that appears during harbor operation. They were initialized with control set to zero (trivial point) or heuristic points, and in all cases the solver converged quickly to high quality local optimal solutions, with no delay on the ships, small stocks and stable flow rates. In many real-life applications we are more interested on finding a local optimum quickly than to spend a lot of time searching for the global optimum.

Test Case 1 is initialized with a trivial point ($u = 0$), therefore not being able to fulfill the shipments in time. This problem was solved in less than one second. The same problem was reformulated as a Mixed Integer Nonlinear Programming (MINLP) model, representing the transfer operation with the mixed-integer approach described in section 3.2. The problem took more than 20

minutes, and was solved with a B&B combined with GRG. It is very likely that other MINLP methods might perform better. Table 2 summarizes the problem, and Figure 3 shows the volumes over the schedule. Table 3 summarizes the other cases.

Table 2 - Test Case 1

#	Parameters	Size	Relax. Problem	Obj.Funct.	Penalty Funct.
1(a)	Tank: 2	Control:48	INITIAL	$8.5*10^5$	7.52
$\Delta t=4$	Vessel: 2	State:56	1	6331.79	7.03
48h	Jetty: 1	Nonlinear:2	2	4.77	5.28
	Pipeline:0	Linear:54	3	0.92	2.61
			4	1.65	0.0
1(b)	Tank: 2	Control:96	INITIAL	$3.9*10^7$	27.17
$\Delta t=4$	Vessel: 2	State:56	1	172.13	2.07
48h	Jetty: 1	Nonlinear:2	2	43.63	1.82
	Pipeline:0	Linear:54	3	100.23	0.45
		Binary: 48	4	3659.87	4.91
			5	52465.82	0.69
			6	175.83	0.69
			7	32.07	0.00

Table 3 - Test cases 2, 3 and 4

#	Parameters	Size	Initial POINT	Solution
2a	Tank: 3	Control:126	$F(x)=6*10^5$	F(x)=7.88
$\Delta t=3$	Vessel: 3	State:92	P(x)=16.34	P(x)=0.07
39h	Jetty: 1	Nonlinear:4	$\mu=0$	$\mu=10^3$
	Pipeline: 1	Linear:88	Trivial	~10s
2b	Tank: 3	Control:126	F(x)=4.07	F(x)=1.31
$\Delta t=3$	Vessel: 3	State:92	P(x)=0	P(x)=0.03
39h	Jetty: 1	Nonlinear:4	$\mu=0$	$\mu=10^2$
	Pipeline: 1	Linear:88	Heuristic	~5s
3a	Tank: 3	Control:196	$F(x)=9*10^5$	F(x)=7.94
$\Delta t=3$	Vessel: 4	State:316	P(x)=16.45	P(x)=0.05
39h	Jetty: 2	Nonlinear:232	$\mu=0$	$\mu=10^4$
	Pipeline: 1	Linear:84	Trivial	~30s
3b	Tank: 3	Control:196	F(x)=4.31	F(x)=4.00
$\Delta t=3$	Vessel: 4	State:316	P(x)=0	P(x)=0.03
39h	Jetty: 2	Nonlinear:232	$\mu=0$	$\mu=10^1$
	Pipeline: 1	Linear:84	Heuristic	~15s
4	Tank: 3	Control:650	F(x)=165.5	F(x)=5.24
$\Delta t=3$	Vessel: 8	State:774	P(x)=10.27	P(x)=0.01
72h	Jetty: 2	Nonlinear:576	$\mu=0$	$\mu=10^2$
	Pipeline: 1	Linear:198	Heuristic	~60s

It is important to notice that cases initialized with the heuristic always found better solutions than the ones initialized with trivial points. Case 4 represents an instance similar to a real-life case, where the resources are very limited compared to the number of vessels to be handled (e.g. 8 vessels in 3 days and only 2 jetties).

4. Conclusion

This article models jetty scheduling problems with the optimal control framework, using controls as flow rates, and continuous variables only. This approach can be extended to other scheduling problems in the process industry. The GRG method fits the problem structure and is able to find local minima with low computational cost. It is possible to find initial feasible points with a heuris-

tic procedure, or by relaxing the problem. For future research, we will investigate the combination of GRG with other methods, such as metaheuristics (similarly to [14]), and the use of the specialized method for optimal control (GRECO) [10] to find better solutions with less computing cost.

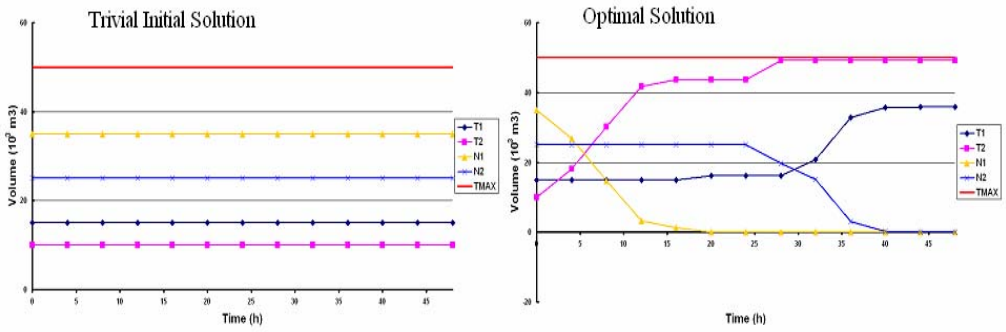


Figure 3 – Volumes at initial point and at optimal solution. (Test Case 1 (a))

References

[1] J. Abadie and J. Carpentier (1969), Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints, in *Optimization*, R. Gletcher Ed., Academic Press, New York.

[2] W.L. Leffler (2000), *Petroleum Refining in Nontechnical Language, 3rd.Ed.*, PennWell, Oakland.

[3] N. Shah, Mathematical Programming Techniques for Crude Oil Scheduling (1996), *Computers and Chemical Engineering*, 20, supplemental issue, S1227 – S1232.

[4] R. Más, and J.M.Pinto (2003), A mixed-integer optimization strategy for oil supply in distribution complexes, *Optimization and Engineering* **4**, No. 1, 23 – 64.

[5] M. Joly, L.F.L Moro, and J.M.Pinto (2002), Planning and Scheduling for Petroleum Refineries using Mathematical Programming, *Brazilian Journal of Chemical Engineering* **19**, No. 2, 207 – 228.

[6] L.M. Simão, D.M. Dias, M.A.C. Pacheco (2007). Refinery Scheduling Optimization using Genetic Algorithms and Cooperative Co-evolution. In: *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI 2007)*, Honolulu.

[7] D.E. Kirk (1970), *Optimal Control Theory: An Introduction*, Prentice-Hall, New York.

[8] J. Abadie (1970), Application of the GRG algorithm to Optimal Control problems, in *Integer and Nonlinear Programming*, J. Abadie Ed., North-Holland, Amsterdam, 191 – 212.

[9] ASTM Standard D-2270 (1991), *Standard Practice for Calculating Viscosity Index from Kinematic Viscosity at 40 and 100ºC*.

[10] J.L.D. Facó (1990), A Generalized Reduced Gradient Algorithm for Solving Large-scale Discrete-time Nonlinear Optimal Control Problems, in *Control Applications of Nonlinear programming and Optimization*, H.B. Siguerdidjane, and P. Bernhard Ed., Pergamon Press, Oxford.

[X] S. Smith and L.S. Lasdon (1992), Solve Large-Sparse Nonlinear Programs using GRG, *ORSA Journal of Computing* **4**, 1 – 15.

[11] Frontline Systems (2006), *Premium Solver Platform Solver DLL Platform Field-Installable Solver Engines User Guide (version 6.0)* [Online]. Available: http://www.solver.com

[12] Fagundez, F.D. (2005), *Modelos do Controle Ótimo de Petróleo e Derivados em Portos*, M.Sc. Dissertation, UFRJ, Rio de Janeiro. [in Portuguese]

[13] Fagundez, F.D., Facó, J.L.D (2006), Jetty Scheduling Optimal Control Models, in *19th International Symposium on Mathematical Programming Abstracts* **1**, 149, UFRJ, Rio de Janeiro.

[14] Z. Ugray, L. Lasdon, J. Plummer, F. Glover, J. Kelly, and R. Marti (2006), Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization, *McCombs Research Paper Series* No. IROM-07-06.

A Decision Support System for Scheduling a Painting Facility in an Automotive Supplier

Jose Pedro García-Sabater, Carlos Andrés Romano,
Julio Juan García-Sabater, Cristóbal Miralles
Universidad Politécnica de Valencia, Camino de Vera, s/n, 46023 Valencia, Spain,
{jpgarcia, candres, jugarsa, cmiralles}@omp.upv.es

This paper deals with the problem of scheduling with setups of two types due to a closed loop configuration with a given quantity of positions. This is typical of closed painting facilities where products are scheduled in different loops. Products are defined by its geometry and its colour. When a change of colour is to be scheduled a setup -horizontal setup- is to be paid (either in terms of cost or in terms of lost capacity). But when in successive loops but in the same position a different geometry is going to be scheduled, a setup cost is also to be paid –vertical setup-. The paper models three variants of the problem. Moreover a greedy heuristic is proposed and it is evaluated through the resolution of a set of problems that covers many aspects of the reality considered.

Keywords: Applications, Production Scheduling, Real World Scheduling.

1. Introduction

Many production systems with closed loop facilities have to deal with the problem of scheduling batches in consecutive loops (e.g. electrolytic painting of automotive parts in closed conveyors, cyclic painting of metallic furniture…). Batch scheduling consists of grouping and scheduling jobs in batches. Such grouping is based on the existence of some similarity between jobs belonging to the same class or family (such as their colour, size or geometry) to minimize changeover or setup time incurred when starting a new batch (where setup time can be sequence dependent or not, see Allahverdi et al. (1999)). The objective of batch scheduling is to minimize the total changeover time or cost satisfying the customer's demand, Mosheiov et al. (2005). In most cases, a job is completed and released only when the batch to which it belongs has finished its processing. But in Just in Time (JIT) environments (such as the automotive sector or supply chain integrated manufacturers), the demand between supplier and customer imposes a cyclic demand, so that batches must be produced within a specific cycle time to minimize the parts and components inventory. Thus, a trade-off between minimizing the total setup time (achieved by reducing the number of batches) and satisfying the customer's demand under JIT perspective (achieved by reducing batch size) must be considered to solve efficiently batch scheduling problem.

Over the past two decades, batch scheduling has become a central topic for researchers, e.g. see Potts and Van Wassenhove, (1992) or Potts and Kovaliov (2000) for a survey. In the one machine problem, Albers and Brucker (1993) showed the complexity of the problem even for the simplest problems and Dobson et al. (1987), Naddef and Santos, (1988), Santos and Magazine (1985) and Webster and Baker, (1995) have defined some procedures for solve this problem in presence of setup times.

A basic assumption in the batching and scheduling problem is that the setup time appears between consecutive batches, but when we have a loop before the machines instead of a linear queue, setup between non consecutive batches can appear, and some concerns related to cyclic scheduling must be considered.

Cyclic scheduling appears in this problem because some times is preferable to repeat a sequence uniform process of supplying to the line by the manpower rather than changing batches in consecutive loop cycles. Cyclic scheduling problems have been studied by several authors. A more complete description of this field can be found in the surveys of Hanen and Munier (1997), Crama (1997) and Gentina (2001). However, these references are closed to their initial scope, and don't consider many additional constraints and situation that might be found in reality.

This paper shows a real productive system which combines cyclic and batching scheduling. The paint line consists of a moving train that forms a continuous loop and contains a fixed number of hollow spaces (positions) that are used to fix the so-called jigs where one or several products are to be hung. The problem of scheduling with setups of two types due to the referred physical configuration arises in real systems. This is typical of closed painting facilities where products are scheduled in different loops. Products are defined by its geometry and its colour. When a change of colour is to be scheduled a setup -horizontal setup- is to be paid (either in terms of cost or in terms of lost capacity). But when in successive loops but in the same position a different geometry is going to be scheduled, a setup cost is also to be paid –vertical setup-. The paper models three variants of the problem. Moreover a greedy heuristic is proposed and it is evaluated through the resolution of a set of problems that covers many aspects of the reality considered.

The remainder of this paper is organized as follows. In Section 2, a detailed description of the real production system will be given. Sections 3 is devoted to give a formal description and a mathematical model for the problem. Some solution procedures are proposed at section 4, and finally the results to implement the procedures at the real productive system are discussed at section 5.

2. Problem description

The problem of cyclic batch scheduling arises in manufacturing facilities with closed conveyors. At the following an example of a closed conveyor facility will be described. Then a broad classification of such systems will be presented. This classification let us to study the different problems that arise in such systems.

An example of closed conveyor facility can be found in a manufacturer of plastic components of varying shapes and colours. Each product is defined by its geometry and its colour. The manufacturer has a closed loop paint line that consists of a moving train that forms that contains a fixed number of hollow spaces so called positions. Products are fixed on each hollow using a special tool so called jig. Each jig might hold a specific and limited number of parts of a given geometry. For the sake of clarity, and without loosing generality we will consider that only one product is hold on each different jig, in real problem this can be achieved by dividing the number of products by the required demand. It is important to note that each model uses a different type of jig, but that the same jig can be used for multiple colours. This means that when the product model to be painted is changed, the jigs must also be changed, but when there is only an alteration in the colour then it is not necessary to change the jigs. A limitation comes with the number of available jigs. Each one has a relevant cost and storing too many of them is not a good option, so usually they are limited. If the colour to be painted in successive units is different, the application of solvent through the pipes that are used to paint might be required. Moreover in some cases, it requires more time than that available between two consecutive positions. In such cases empty positions have to be allowed between those two batches.

In some cases the PLC controlling the painting pipes need to be reset if different geometries have to be painted. Such changeover might require a quantity of time superior to that available between two consecutive positions. Therefore when a change of colour is to be scheduled a setup -horizontal setup- is to be paid (either in terms of cost or in terms of lost capacity). When a change of geometry is to be scheduled a setup cost might be paid in terms of lost capacity. But, an here is the novelty, when in successive loops but in the same position a different geometry is going to be scheduled, a setup cost is also to be paid (vertical setup).

The parts pass continuously pass through a painting area located at a fixed position on the line. In the case studied, batching scheduling is desirable to minimize setup time between consecutive batches of similar parts and giving good response to the customer, and cyclic scheduling is desirable to reduce work in progress inventory between the facility and the automotive customers.

Summarizing each type of jig supports a defined number of parts (in our case 1 is general enough). In the system, different quantities of jigs exist for each model and it is not possible to exceed the maximum number of jigs per model daily.

Two types of setups could be considered: Horizontal Setups and Vertical Setups

1) **Horizontal Setups:** are conventional changeovers between batches of consecutive batches. They are related to two types of changes geometry changes and colour changes. Geometry changes are required for instance when the software controlling the pipes need to be updated. Colour changes might require solvent to clean pipes or space between two colours.
2) **Vertical Setups:** as previously mentioned, when a geometry change is required in the same position but in the following loop, a jig change must be carried out. It is necessary to use worker capacity to do this change. The use of a worker has associated a cost and this is the reason why every geometry change has associated a setup cost.

Different situations might arise depending on the empty space that is required to be schedule between consecutive batches. The situation 1 happens when the conveyor moves slowly enough it is possible to do the changes without losing capacity. This means that it is possible to replace the colours or the jigs in a position when it remains at the same place. However, some times the cycle time of the conveyor is faster and it is not possible to perform a change in a specific position into the cycle time. Situation 2 and 3 arise when empty spaces have to be allowed between positions. Situation 2 appears when only one position is to be freed either for geometry change and for colour change. Situation 3 requires different quantity of empty positions depending if the changeover is due to a geometry changeover or a colour changeover. Loosing capacity due to the changeovers is specially relevant in this situations, because due to the continuous movement of the loop and the limited capacity of the workers to carry out such changes, the number of jig changes during each loop still remains limited.

In Figure 1, it can be seen that the change between consecutive blocks of different models makes it necessary to include empty jigs in the sequence. This setup is necessary if consecutive blocks have different model types. When carrying out a model change, one hollow in the first block must be empty and two hollows in the next block must also be empty (see Figure 1). If a product is going to be scheduled on a block which in the previous loop contained a product with a different model type, then the jigs must also be changed but usually without loose capacity.

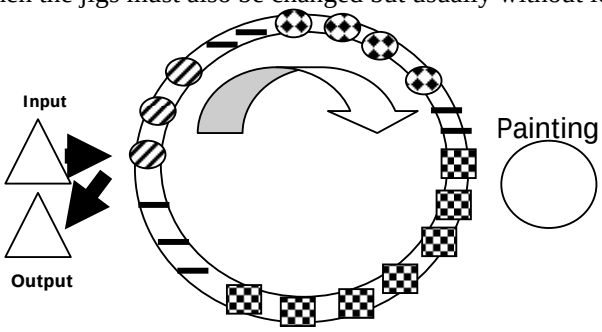


Figure 1. System layout

3. Problem statement.

3.1. Case 1. Problem description.

A better description can be stated using figure 2. The figure shows three cycles or loops with a number of different geometries (G1, G2, G3) colours (C1, C2,C3) in each loop. There are setup costs due to jig changes in consecutive loops and setup cost due to color in consecutive positions.

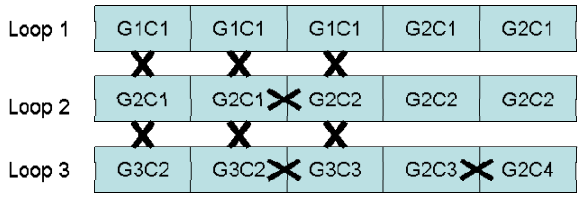


Figure 2. Schedules for Situation 1 at successive loops

The problem can be stated as follows: Sequence a set of products (defined by its geometry and its colour) in different quantities each one. Knowing that on each position only one unit can be scheduled. The number of units of a given model (in any set of BLOCS consecutive positions is limited by the availability of jigs to hang them. We have to minimize the number of colour changes (considering consecutive units) because each one has a cost (mainly in solvents), try to minimize the number of jig changes, between the same place of consecutive loops (mainly worforce cost), and try to minimize the number of empty positions (mainly energy cost). In this problem the geometry changeover in consecutive units is not relevant.

3.2. Case 2. Problem description.

The main difference between this case and the previous one is that when a geometry or colour change is scheduled an empty position should be scheduled. Note that we are considering consecutive positions and no consecutive loops. The objective will be then to sequence a set of products (defined by its geometry and its colour) in different quantities each one. At each position only one unit can be scheduled. The number of units of a given geometry (in any set of BLOCS consecutive positions is limited by the availability of jigs to hang themI. When a change in colour or model is to be scheduled an empty position has to be placed between those sets of units. We have to minimize the number of colour changes (between consecutive units) because each one has a cost (mainly in solvents), and to minimize the number of jig changes, the same position of consecutive loops (mainly worforce cost), and minimize the number of empty positions (mainly energy cost).

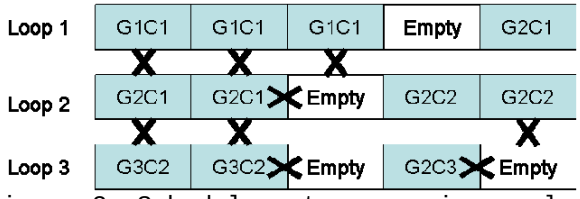


Figure 3. Schedules at successive cycles

A better description can be stated using figure 3. The figure shows three cycles or loops with a number of different geometries (G1, G2, G3) colours (C1, C2,C3) in each loop. There are capacity looses (one position) between consecutive colour changes, and setup costs due to jig changes.

3.3. Case 3. Problem description.

This case is a variant that arises when the geometry change needs a number of consecutive empty positions and that number is different considering colour changes.

Figure 4 tries to depict the problem. The figure shows three cycles or loops with a number of different colours and geometries in each loop. There are capacity loose (one position) between consecutive colour changes and geometry changes. Also jig change costs are considered.

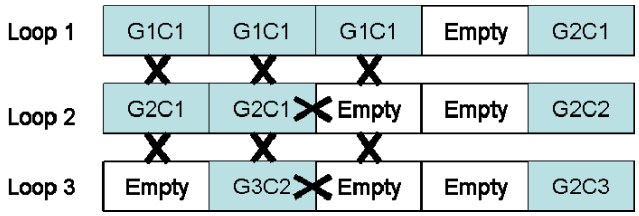


Figure 4. Schedules at successive cycles

3.4. General Problem definition

In every case, the problem may be stated as follows: given a set of products to manufacture (defined by their geometry, colour and required production quantity), the number of positions per loop (the system capacity), and the jig availability the objective is to schedule the products in the minimum number of blocks reducing the total cost given by setup cost (either horizontal and vertical) and the empty positions.

The objective function in this problem is to minimize the total number of empty positions in the blocks or maximize the capacity of the facility.

4. Solution procedures

The 1, 2 and 3 cases reflect variants of the same problem: to schedule the production taking into account not only the previously scheduled product but also the geometry of the product scheduled on the previous loop. Some herefore heuristic algorithms have been developed. When working with the real versions of the problem some questions arised that contributed to define the algorithms. The most important is that to maintain the previous loop geometry sequence is a very relevant point because is the only aspect that was considered by the workforce. Moreover “if every product is to be manufactured, if the geometry can be maintained, there is no need to change it” (as a foreman said). Of course the view from the board of directors point of view is slightly different because more gaps imply less productivity.

The following algorithm has been develop to solve the previous problems. It has to be noted that the following algorithms work for the third case (and therefore for the first and second).

The notation used for the models is kept but some new is added:

i is an index for products

[j]: represents the product sequenced at Position j

K[j] represents the colour of the product at Position j

H[j] represents the geometry of the product at Position j

R[j] represents the remaining quantity of units to be scheduled of product sequenced at position j

List is a list including the Code of the products to be scheduled. K is the index that will cover the ordered List. List(k) will be the product in place k. H(List(k)) is the Geometry of that product. K(List(k)) is the colour of that product. R(List(k)) is the remaining quantity of units of product List(k) to be scheduled. A product is to be removed if R(List(k))=0. Initially R(List(k))=Q(List(k))

JA(h) is the availability of Jigs for Geometry h

JU(h) is the Usage of Jigs for Geometry h during the previous BPL-1 positions

LastColour is the colour of the last scheduled product.

The implemented procedure is the following one

Step 1: List is ordered according to the chosen criteria.

Step 2: Initialize

Step 2.1 j:=1; k:=1

Step 2.2 JU(h)=0 for every h

Step 4: if **List** is empty then Finish

Step 5: ik k=0 then **Insert a Null Product**
Step 5.1 [j]:=0;
Step 5.2 j:=j+1
Step 5.3 JU(H([j-BLOCS]):= JU(H([j-BLOCS])-1
Step 5.4 Go to 8
Step 6: Insert the k-th product in the List
Step 6.1: [j]:=List[k]; LastColour:=K[j]
Step 6.2: R[j]=R[j]-1;
Step 6.3: JU(H[j]):= JU(H[j])+1;
Step 6.4: JU(H[j-BPL]):= JU(H[j-BPL])-1;
Step 6.5: j=j+1
Step 6.6: Go to 7
Step 7: Check if [j-1] can be reinserted
Step 7.1: if R[j]=0 then Remove [j] from the **List ;** Go to 8
Step 7.2: if JU(H(List(k)))=JA(H(List(k))) then Go to 8
Step 7.3: Go to 6
Step 8: Check if a Product with the same geometry H[j-BLOCS] can be inserted
Step 8.1: Count the number of empty positions before the position [j]
Step 8.2: if the number of empty positions is less than EH a change in model might be scheduled otherwise go to Step 8.4
Step 8.3: Look for the first Product in **List** with the same Geometry of H[j-BLOCS] and LastColour as colour. If the product exists set k to the place where it can be found and go to 5.
Step 8.4: if the number of empty positions is less than EK a change in colour might be scheduled otherwise go to Step 9
Step 8.5: Look for the first Product in **List** with the same Geometry of H[j-BLOCS]. If the product exists set k to the place where it can be found and go to 6.
Step 9: Look for the first Product in List to be inserted.
Step 9.1: Count the number of empty positions before the position [j]
Step 9.2: if the number of empty positions is less than EH a change in model might be scheduled otherwise go to Step 9.4
Step 9.3: Look for the first Product in **List** where LastColour is its colour and JU(H(List(k)))<JA(H(List(k))). If the product exists set k to the place where it can be found and go to 5.
Step 9.4: if the number of empty positions is less than EK a change in colour might be scheduled otherwise go to Step 9.7
Step 9.5 : Look for the first product in List where JU(H(List(k)))<JA(H(List(k))).
Step 9.6: If such a product has been found set k to the place where it is located in **List**; go to Step 6
Step 9.7: If the end of the list is reached then k:=0; go to Step 4.

With respect to the previous algorithm several remarks are to be noted. First, this algorithm tries to reduce intrinsically the number of jig changes (Step 8). The reason for that come from the experienced reality. Although for the company the main direct costs to be considered is the cost of a colour change and the excess of empty blocks, the only real effect over the manpower of a given schedule is the number of jig changes that they have to perform. Therefore a schedule improving the colour change level but having too many jig changes is never going to be understood neither by the workers nor by the foreman. Therefore if a jig can be maintained it should be maintained, except in the case that the same product is to be scheduled. Second, the jig limitation is considered

with variables JU in Steps 7.2 and 9.5. Obviously it is not necessary to keep control of that if the geometry of the previous loop is to be repeated (Step 8). Third, for the first loop (j<BLOCS) some of the operations do not apply but for simplicity those controls have been avoided. Fourth, it is assumed that the colour change needs more empty positions than the model change. Our experience in reality never showed the opposite situation. Fifth, there is a variant (that has been tested) that always leaves empty the last two positions of a given loop. This variant intended to allow that in the following loop the same set of jigs could be placed, reducing therefore the number of jig changes. This variant only requires small changes mainly adding a step between 4 and 5, checking if the end of the loop has been reached and if so, include two empty positions and move to step 8. It has not been added on the previous algorithm for the sake of clarity.

5. Computational experiences.

As has been described, the problem presented is an innovative problem that can be found in different industrial environments. As there is no standard set of benchmark instances for testing solving procedures, a two-level five factors full factorial experimental study was constructed. The particular case that has been described includes several characteristics that have been taken as inspiration to generate this benchmark. In this sense every problem always has 1600 units to be grouped in batches and scheduled, but depending on the problem we have different number of geometries and colours involved. This is like saying that the daily demand is about 1600 positions what can be considered as an appropriated order of magnitude.

The problems were generated according to the following two factors: the number of geometries (NrGeometries) and the number of colours (NrColours).

For the first factor two levels were defined, High (40 geometries) and Low (20 geometries). On the other hand, for the second factor there were problems with High number of colours (20) and Low number of colours (10). For each kind of problem 40 instances were generated, so that the definitive benchmark is composed of 160 problems. All of them were run with the different variants of the heuristic algorithm described in section 4.

There are some parameters related to the physical configuration of the system. The two that are relevant for us are the number of positions per loop and the number of jigs per geometry. Together with the capacity of the systems (the 1600 units per day considered before) those are decisions that affect to the problem and the resolution procedure: Number of positions per loop (100-30), maximum percentage of geometries per loop (75 % of a loop can hold the same geometry, 25% of a loop can hold the same geometry), Criteria for sorting the batches (Sorting the list by the most demanded colour or Sorting the list by the most demanded geometry).

The 160 problems generated and they were run for the 4 variants of the system configuration and for the 4 variants of the heuristic, what supposes 2560 experiments. As it is not possible to have an optimal solution for the benchmark, in order to extract valid conclusions about which variant is more efficient for every kind of problem, the indicator used was the Relative Perceptual Deviation (RPD) for every experiment defined as:

$$RPD_{\exp} = \frac{Sol_{\exp} - BestSol_{ins}}{BestSol_{ins}} \cdot 100$$

From our ANOVA analysis we may summarize the main conclusions obtained by means of Fisher Test Graphics. The main statistically significant factors were the strategy and the direction, being indifferent the criteria applied both for selection and even results rules:

It is interesting to remark that the different problems have been built considering that the total demand is constant and varying the number of geometries and colors. But the real problem has another two aspects that are defined in reality: the length of the loop and the quantity of available jigs. Comparing the two ways of ordering the list (by color or by model) we can see that the difference between the two sorting methods for the RPD is favourable to ordering by colour, but not with a big difference (less than 30%). On the other hand the values of JPD have a bigger variance

(nearly 100%) between both ordering methods. We can conclude that if the focus is in reducing Jig Changes there is one ordering method much better than the other (in our case ordering by geometry). If we try to measure the effect of leaving to empty positions at the end of the loop we can see that although it halves the quantity of jig changes, the number of empty blocks is strongly affected.

If we have to assess about the length of a loop again both metrics JPD and RPD had been checked. Figure 9 show that there is a strong relation between the length of the loop and the number of empty positions, as larger that number less overall empty positions. Data of Figure 10 shows that the number of jig changes is hardly affected by the length of the loop. If comparing the relation between the Jig availability and the quantity of different geometries we can easily see that the Jig availability is relevant only to the number of empty spaces if the number of different geometries is Low. If the number of geometries is low the different procedures are more relevant to the overall solutions for both the number of empty positions and the number of jig changes. By analysing the results we can say that ordering the products considering the most demanded model and leaving 2 empty positions at the end of the loop improves significantly the number of jig changes, but worsens the number of empty positions.

Considering the problem with a real perspective we have to make two additional considerations. First, the number of empty jigs and the number of jig changes is relevant considered against the total length of the schedule. Second, the number of colour changes is an absolute value that has to be taken into account. Having a look on the first aspect the relative extra time (i.e how much time would we have to work due to empty positions) moves around a 10% between the worst schedule and the best one. But if we have a look on the relative extra time that the workers will have to work changing jigs that relation (the worst against the best) is about a 35%. The number of total color changes is about 5 times worse between the best obtained solution and the worst one.

6. Conclusions

In this paper we have addressed a scheduling problem that can be considered innovative in the field of scheduling with setups. It does take into account two different types of changeovers, those due to two consecutive batches, that has been called (horizontal setup) and those due to the relation between consecutive loops in the same position. The objective is to schedule minimizing the cost of both types of setups. The cost of the setup is related to effective cost (either solvent or workforce) but also is an opportunity cost of lost capacity. The case treated in this paper does not fall into any category of the known scheduling problems in the literature, but it bears certain similarities with cyclic scheduling, batch scheduling or flowshop sequence dependent scheduling. Three different cases have been presented, each one incorporated a new feature. A simple direct heuristic has been developed with two variants, considering two alternatives for each variant. The heuristic has been used to analyze the effects of the values of the problem (such as number of geometries and colors), and the physical configuration of the system (such as the length of the loop and the quantity of available jigs for the same geometry).

Acknowledgments

This work was developed under research projects GESCOFLOW (DPI2004-02598) supported by the Spanish National Science&Technology Commission CICYT

References

[1] A. Allahverdi, J.N.D. Gupta, T. Aldowaisan, (1999), A review of scheduling research involving setup considerations, *Omega* **27**, 219 – 239.

[2] A. Allahverdi, C.T. Ng, , T.C.E. Cheng, and M.Y. Kovalyov (2007). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*. In press.

[3] G. Mosheiov, D. Oron, Y. Ritov (2005), Minimizing flow-time on a single machine with integer batch sizes, Operations Research Letters **33**, 497 – 501.

[4] C.N. Potts and L.N. Van Wassenhove, (1992), Integrating scheduling and batching lot-sizing a review of algorithms and complexity, Journal of Operational Research Society **43**, 395 – 406.
[5] C.N. Potts and M. Kovalyov (2000), Scheduling with batching a review, *European Journal of Operational Research* **120**, 228 – 249.
[6] S. Albers and P. Brucker (1993), The complexity of one-machine batching problems, *Discrete Applied Mathematics* **47**, 87 – 107.
[7] G. Dobson, U.S. Karmarkar and J.L. Rummel, (1987), Batching to minimize flow times on one machine, Management Science **33**, 784 – 799.
[8] D. Naddef and C. Santos (1988), One-pass batching algorithms for the one-machine problem, Discrete Applied Mathematics **21**, 133 – 145.
[9] C. Santos and M. Magazine, (1985), Batching single operation manufacturing systems, Operations Research Letters **4**, 99 – 103.
[10] S. Webster and K.R. Baker (1995), Scheduling groups of jobs on a single machine, Operations Research. **43**, 692 – 703.
[11] C. Hanen, A. Munier, (1997), Cyclic scheduling on parallel processors: an overview. In: Chretienne, P., Cofman, E.G., Lenstra, J.K., Liu, Z. (Eds.), Scheduling Theory and its Applications. John Wiley & Sons Ltd
[12] Y. Crama, (1997), Combinatorial optimisation models for production scheduling in automated manufacturing systems, European Journal of Operational Research **99**, 136 – 153,
[13] J.C. Gentina, O. Korbaa and H. Camus, (2001). Problèmes d'ordonnancement cyclique. In: P. Lopez, F. Roubellat (Eds.), (Chapter 7)**,** Ordonnancement de la Production, Hermès Science, IC2 Productique, 197 – 223.

The Two-Stage Assembly Flow Shop Scheduling with an Availability Constraint

Hatem Hadda
Unité de Recherche URAII, INSAT, Centre Urbain Nord B.P. N°676, 1080 Tunis, hatem.hadda@esti.rnu.tn
Najoua Dridi
Unité de Recherche OASIS, ENIT, B.P. N°37, le belvédère 100 Tunis, najoua.dridi@enit.rnu.tn
Sonia Hajri-Gabouj
Unité de Recherche URAII, INSAT,Centre Urbain Nord B.P. N°676, 1080 Tunis, sonia.gabouj@insat.rnu.tn

In this paper, we deal with the two-stage assembly flow shop scheduling problem with an availability constraint. The objective is to minimize the maximum completion time of the jobs (i.e. makespan). We are interested in the nonresumable model. We provide two heuristics with tight worst-case ratio bound of 2 when the availability constraint is imposed on one of the first stage's machines. For the case where the gap is imposed on the assembly machine we show that no polynomial heuristic algorithm with finite error bound can be found.

Keywords: Machine Scheduling, assembly flow shop, availability constraint, worst-case analysis.

1. Introduction

While most papers dealing with the scheduling field assume that the machines are available all the time, a growing number of works are considering that the machines are subject to breakdown; authors differentiate between the *probabilistic* case where breakdowns are random and the *deterministic* case where the date of unavailability period and its duration are known. Different models are distinguished, the *semiresumable* (*sr*) case that if a job cannot be finished before the next down period of a machine then the job will have to partially restart when the machine becomes available again. This model contains two special cases: namely the *resumable* (*r*) when the job can be continued without any penalty and the *nonresumable* (*nr*) case when the job needs to totally restart.
The availability constraint was considered for many scheduling problems. The one machine problem was tackled by [1,6,17,18,21], the parallel machines by [13,19]. For the flow shop problem Lee [14] considered the two-machine flow shop problem with a resumable availability constraint in one of the two machines, he proved that the problem is NP-hard in the ordinary sense and developed a pseudo-polynomial dynamic programming algorithm to solve the problem. He also provided two heuristics with their error bound analysis. Later, Lee [15] studied the same problem but within the semiresumable case and availability constraints in both machines. He provided complexity analysis, developed a pseudo-polynomial dynamic programming and also proposed heuristic algorithms with an error bound analysis. Cheng and Wang [7] provided an improved approximation algorithm, for the case where the unavailability period is on the first machine; this algorithm has a relative worst-case error bound of $4/3$. Later, Breit [5] presented for the problem with one unavailability constraint on the second machine, an improved algorithm with a relative worst-case error bound of $5/4$. The two-machine flow shop problem with many unavailability periods was also studied by [4,12]. The general *m* machine flow shop problem was considered by Aggoune [2], for which he proposed a heuristic approach based on a genetic algorithm and a tabu search. More recently, Allaoui and Artiba [3] investigated the two-stage hybrid flow shop scheduling problem, with only one machine on the first stage and *m* machines on the second stage, the authors discussed the complexity of the problem, gave a branch and bound model and calculated the worst-case performances of three heuristics.
In this paper, we deal with the two-stage assembly flow shop scheduling with *nonresumable* availability constraint in one of the *m*+1 machines. The objective is to minimize the maximum comple-

tion time of the jobs, i.e., the makespan. We will use the notation $Am|nr-a(M_k)|C_{\max}$ to denote this problem, where M_k designates the machine subject to the availability constraint.

When considering the problem $Am\|C_{\max}$ (without any machine availability constraint), we obtain the known Two-Stage Assembly Flow Shop problem. This problem can be viewed as an extension of the two-machine flow shop problem. The assembly problem was first introduced by Lee and al. [16] who studied the 3-Machines assembly flow shop (3MAF), where there is two machines in the first stage and a unique assembly machine in the second stage. The objective is to minimize the makespan. The problem was shown to be NP-hard in the strong sense and a number of polynomial cases was identified. Authors also proposed a branch and bound algorithm and three heuristics with their error bound analysis. Potts and al. [20] proposed a generalization to 3MAF problem by considering more than two machines at the first stage, which they noted $Am\|C_{\max}$. The authors presented a complexity study and proposed a number of heuristics with a study of their relative and absolute errors.

Hariri and Potts [10] proposed a set of elimination rules which they used in a branch and bound algorithm, they were able to solve big size problems. More recently, Xi and al. [22] presented, for the 3MAF, a series of heuristics based on the ideas of Johnson [11] and Gupta [8], they also gave a worst-case analysis.

In addition, Haouari and Daouas [9] dealt with the inverse problem of 3MAF, they called this problem the 3-machine dismantling flow shop problem (3MDF) and they showed that this problem is equivalent to the original problem (3MAF). The authors also proposed an efficient branch and bound algorithm.

This paper is organized as follows. After presenting the notations and the problem description, we discuss in section 3 the configuration where the unavailability period is on one of the first stage‘s machines. We namely develop heuristic methods with an error bound analysis. In section 4, we investigate the case where the gap is on the assembly machine, a worst-case analysis is also provided.

2. Notations and problem description

For defining formally the problem, let us define:

n	total number of jobs to be scheduled;
J_i	job *i*, $i \in \{1,..,n\}$;
J	set of all the jobs: $J = \{J_1, J_2, ..., J_n\}$;
M_A	assembly machine;
M_k	one of the first stage's machines, $k \in \{1,..,m\}$;
p_{ik}	processing time of job *i* on M_k, $k \in \{A,1,..,m\}$;
$\pi = (\pi_1, ..., \pi_n)$	a schedule of *n* jobs;
$C_{\max}(\pi)$	makespan of the schedule π;
s_k	start time of the unavailability period on M_k, $k \in \{A,1,..,m\}$;
t_k	finish time of the unavailability period on M_k, $k \in \{A,1,..,m\}$;
$C^*_{\max}$	makespan of the optimal solution;
J_k^{π}	the crossover job of M_k under π.

The Two-Stage Assembly Flow Shop problem ($Am\|C_{\max}$) is given by a set of *n* available jobs to be processed on a set of (*m*+1) machines. Each job *i* consists of *m*+1 operations {O_{i1}, O_{i2}, …, O_{im},O_{iA}} to be processed respectively on M_k $k \in \{1,..,\mathrm{m}\}$ and M_A. The operation O_{iA} cannot begin

until all operation O_{ik} $k \in \{1,..,m\}$ are finished. Besides, the n jobs are simultaneously available at time 0, each machine can handle no more than one job at a time.
We assume that one of the machines M_k $k \in \{A,1,..,m\}$ is unavailable during the period from s_k to t_k ($0 \le s_k \le t_k$). If a job cannot be finished before the unavailability period (the gap) of a machine then the job has to restart. This case is known as *nonresumable*. Hence the problem will be denoted by $Am|nr-a(M_k)|C_{\max}$. The problem is to find the schedule that minimizes the total completion date of the last job visiting the machines M_A (i.e. the makespan $C_{\max}(\pi)$).
For a given schedule π, let J_k^{π} be the first job that finished after t_k on M_k. We call J_k^{π} the crossover job of M_k under π.
As the $Am||C_{\max}$ is NP-hard in the strong sense [16,20], then our more general problem $Am|nr-a(M_k)|C_{\max}$ is also strongly NP-hard. We note that the only variant of this problem solved in polynomial time is the well-known $F2||C_{\max}$ problem solved by Johnson [11].

3. Availability constraint on machine $M_h, 1 \le h \le m$: $Am|nr-a(M_h)|C_{\max}$

In this configuration, only the machine M_h for a fixed $1 \le h \le m$ is unavailable during the period from s_h to t_h.

3.1. Basic proprieties

We first point out that if $\sum_{1 \le i \le n} p_{ih} \le s_h$, then the availability constraint does not affect the schedule, hence we assume that $\sum_{1 \le i \le n} p_{ih} > s_h$ and thus the completion time on M_h for any schedule will be greater then $\sum_{1 \le i \le n} p_{ih} + t_h - s_h$, hence:

$$\sum_{1 \le i \le n} p_{ih} + t_h - s_h \le C_{\max}^* \quad (1)$$

For a given sequence of jobs π on M_h, there would be two sets of jobs, those which will be finished on M_h before the s_h (called the set X_π) and those which will be finished after the t_h (called the set Y_π). We have $\sum_{i \in X} p_{ih} \le s_h$, we then note $\Delta_\pi = s_h - \sum_{i \in X} p_{ih}$. (We will drop the reference to the sequence π whenever no confusion can arise).
We also remark that for a given schedule π, and if we note $J_{i_0} = J_h^{\pi}$ then :

$$p_{i_0 h} > \Delta \quad (2)$$

Otherwise, J_{i_0} will be finished before s_h on M_h.

Note that it is sufficient to consider the *permutation solution* for $Am||C_{\max}$ [20]. However, if an availability constraint is imposed in one of the first stage's machines, then the following result is verified.

Lemma 1
It is possible that there does not exist any permutation solution to be optimal for $Am|nr-a(M_h)|C_{\max}$.

Proof:

Consider an instance with: *m*=2, *n*=2 and $p_{11}=4$, $p_{12}=11$, $p_{21}=5$, $p_{22}=1$ and $p_{1A}=p_{2A}=2$. We have also $s_1=4$, $t_1=5$.

It can be checked that the unique optimal solution is to sequence J_1J_2 on M_1, and J_2J_1 on M_2 and M_A. With makespan equal to 14. □

Although we cannot restrict the research to the permutation solutions, the following result gives us a considerable space reduction.

Lemma 2

For $Am|nr-a(M_h)|C_{\max}$, $1\le h\le m$. There exists an optimal solution where there is the same sequence of jobs on the machines M_k, $k\neq h$ and M_A.

Proof:

Suppose that there is an optimal solution π where the sequence of jobs on a machine M_k for $k\neq h$ is different of the sequence on M_A. Then there exists two jobs J_i and J_j such that J_j is immediately sequenced before J_i on M_k. But J_j follows J_i on M_A.

It can be checked that the inversion of the order of J_i and J_j on M_k will not lead to a higher makespan. And so by applying the same inversion to all the jobs that appears in the wrong order, we can reorder all jobs to satisfy condition of lemma 2. □

Remark 1: Lemma 2 allows us to reduce the space size from $(n!)^{m+1}$ to $(n!)^2$. From here we will consider only this family of schedules. For notational convenience, and for a given schedule π, the sequence of jobs on the machines M_k, $k\neq h$ and M_A will be denoted simply by π, and the sequence of jobs on the machine M_h will be denoted by π^h.

We present now the worst case analysis.

3.2. Arbitrary heuristic worst case performance

The following theorem gives the bound for a heuristic *H* giving an arbitrary solution. We note that in such a solution no machine can be idle while a job is ready to be processed on that machine.

Theorem 1

For any heuristic *H* producing an arbitrary schedule π for $Am|nr-a(M_h)|C_{\max}$, we have $C_{\max}(\pi)\le 3C^*_{\max}$.

Proof:

Two configurations are to consider:

Case 1: the gap is not part of the critical path, then there exist $1\le j\le n$ and $1\le k\le m$ such that

$$C_{\max}(\pi)=\sum_{i=1}^{j}p_{\pi_i k}+\sum_{i=j}^{n}p_{\pi_i A}\le\sum_{1\le i\le n}p_{ik}+\sum_{1\le i\le n}p_{iA}\le 2C^*_{\max}$$

Case 2: the gap is a part of the critical path, then there exists $1\le j\le n$ and $1\le j'\le n$ such that:

$$C_{\max}(\pi)=\sum_{i=1}^{j}p_{\pi_i^h h}+t_h-s_h+\Delta+\sum_{i=j'}^{n}p_{\pi_i A}\le\sum_{1\le i\le n}p_{ih}+t_h-s_h+\Delta+\sum_{1\le i\le n}p_{iA}$$

Let's note $J_{i_0}=J_h^{\pi}$. From (1) and (2) we obtain $C_{\max}(\pi_H)\le 2C^*_{\max}+p_{i_0 h}\le 3C^*_{\max}$. □

We now present two heuristics that guaranties a bound of 2.

3.3. H_0 heuristic worst case performance

In the heuristic H_0 we sort the set of jobs according to the processing times on the machine M_h in non-decreasing order. We then consider the resultant permutation solution.

Theorem 2

If we apply the heuristic H_0 to sequence jobs for $Am|nr-a(M_h)|C_{\max}$ then $C_{\max}(\pi) \le 2C^*_{\max}$ and the bound is tight, where π is the permutation given by the heuristic H_0.

Proof:

If the gap is not a part of the critical path then, and as explained in the proof of the previous theorem, we have $C_{\max}(\pi) \le 2C^*_{\max}$.

Otherwise, if the gap is a part of the critical path, then there exists $1 \le j \le n$ such that:

$$C_{\max}(\pi) = \sum_{i=1}^{j} p_{\pi_i h} + t_h - s_h + \Delta + \sum_{i=j}^{n} p_{\pi_i A} \quad (3)$$

<u>*Case 1:*</u> if $j = n$ then (3) gives

$$C_{\max}(\pi) = \sum_{1 \le i \le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_n A} \quad (4)$$

Let's note $J_{i_1} = J_h^{\pi}$, we know from (2) that $p_{i_1 h} > \Delta$, as we have applied the heuristic H_0 then $p_{\pi_n h} \ge p_{i_1 h} > \Delta$. As $p_{\pi_n h} + p_{\pi_n A} \le C^*_{\max}$ then we conclude the following from (1) and (4)

$$C_{\max}(\pi) = \sum_{1 \le i \le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_n A} \le (\sum_{1 \le i \le n} p_{ih} + t_h - s_h) + (p_{\pi_n h} + p_{\pi_n A}) \le 2C^*_{\max}$$

<u>*Case 2:*</u> If $j \ne n$ then we have $p_{\pi_n h} \ge p_{i_1 h} > \Delta$ and from (3) we get

$$C_{\max}(\pi) = \sum_{i=1}^{j} p_{\pi_i h} + t_h - s_h + \Delta + \sum_{i=j}^{n} p_{\pi_i A} \le \sum_{i=1}^{j} p_{\pi_i h} + p_{\pi_n h} + t_h - s_h + \sum_{i=j}^{n} p_{\pi_i A}$$

Then $$C_{\max}(\pi) \le \sum_{1 \le i \le n} p_{ih} + t_h - s_h + \sum_{1 \le i \le n} p_{iA} \le 2C^*_{\max}$$

In conclusion $C_{\max}(\pi) \le 2C^*_{\max}$.

The following example shows that the bound is tight. Consider an instance with *m*=1(i.e. a two machine flow shop problem), *n*=2, $p_{11} = 1$, $p_{21} = z$, $p_{1A} = 2$ and $p_{2A} = 2$. we also have s_1=*z* and t_1=*z*+1, where *z* is a positive integer.

If we apply H_0 to the problem we get a sequence J_1, J_2 with $C_{\max}(\pi) = 2z + 3$. However, the optimal solution is to sequence J_2 followed by J_1 with $C^*_{\max} = z + 4$. Hence $C_{\max}(\pi) / C^*_{\max}$ approaches 2 as *z* approaches infinity. □

3.4. H_1 heuristic worst case performance

In the heuristic H_1 we apply Johnson's algorithm (*JA*) to sequence the *n* jobs by considering only the processing times on the machines M_h and M_A. The optimal permutation obtained for the two machines flow shop problem is then applied for the $Am|nr-a(M_h)|C_{\max}$ problem.

Theorem 3

If we apply the heuristic H_1 to sequence jobs for $Am|nr-a(M_h)|C_{\max}$ then $C_{\max}(\pi) \le 2C^*_{\max}$ and the bound is tight, where π is the permutation given by the heuristic H_1.

Proof:

If the gap is not a part of the critical path then, and as explained previously we have $C_{\max}(\pi) \le 2C^*_{\max}$.

Otherwise, if the gap is a part of the critical path, then there exists $1 \le j \le n$ such that:

$$C_{\max}(\pi) = \sum_{i=1}^{j} p_{\pi_i h} + t_h - s_h + \Delta + \sum_{i=j}^{n} p_{\pi_i A} \quad (5)$$

Case 1: if $j = n$ then (5) gives

$$C_{\max}(\pi) = \sum_{1\le i\le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_n A} \tag{6}$$

Let's note $J_{i_1} = J_h^{\pi}$, we know from (2) that $p_{i_1 h} > \Delta$. As π_n follows J_{i_1} in the sequence then: $\min(p_{i_1 h}, p_{\pi_n A}) \le \min(p_{i_1 A}, p_{\pi_n h})$.

- If $\min(p_{i_1 h}, p_{\pi_n A}) = p_{i_1 h}$ then $p_{\pi_n h} \ge p_{i_1 h} > \Delta$ so

$$C_{\max}(\pi) = \sum_{1\le i\le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_n A} \le (\sum_{1\le i\le n} p_{ih} + t_h - s_h) + (p_{\pi_n h} + p_{\pi_n A}) \le 2C^*_{\max}$$

- If $\min(p_{i_1 h}, p_{\pi_n A}) = p_{\pi_n A}$ then $p_{\pi_n A} \le p_{i_1 A}$ so

$$C_{\max}(\pi) = \sum_{1\le i\le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_n A} \le (\sum_{1\le i\le n} p_{ih} + t_h - s_h) + (p_{i_1 h} + p_{i_1 A}) \le 2C^*_{\max}$$

Case 2: if $j \ne n$ then two sub-cases are to consider

Case 2-1: there exists a job $i_2 = \pi_{j'}$ for $j' > j$ such that $p_{i_2 h} \ge p_{i_1 h}$ then $p_{i_2 h} \ge \Delta$. From (5) we get:

$$C_{\max}(\pi) = \sum_{i=1}^{j} p_{\pi_i h} + t_h - s_h + \Delta + \sum_{i=j}^{n} p_{\pi_i A} \le \sum_{i=1}^{j} p_{\pi_i h} + p_{i_2 h} + t_h - s_h + \sum_{i=j}^{n} p_{\pi_i A}$$

Then $$C_{\max}(\pi) \le \sum_{1\le i\le n} p_{ih} + t_h - s_h + \sum_{1\le i\le n} p_{iA} \le 2C^*_{\max}$$

Case 2-2: for any job $i_2 = \pi_{j'}$ such that $j' > j$ we have $p_{i_2 h} < p_{i_1 h}$. As J_{i_2} follows J_{i_1} in the sequence then: $\min(p_{i_1 h}, p_{i_2 A}) \le \min(p_{i_1 A}, p_{i_2 h})$. So necessarily $\min(p_{i_1 h}, p_{i_2 A}) = p_{i_2 A}$ and subsequently $p_{i_2 A} \le p_{i_2 h}$. We can then write

$$\sum_{i=j}^{n} p_{\pi_i A} \le p_{\pi_j A} + \sum_{i=j+1}^{n} p_{\pi_i A} \le p_{\pi_j A} + \sum_{i=j+1}^{n} p_{\pi_i h} \tag{7}$$

From (5) and (7) we get

$$C_{\max}(\pi) = \sum_{i=1}^{j} p_{\pi_i h} + t_h - s_h + \Delta + \sum_{i=j}^{n} p_{\pi_i A} \le \sum_{i=1}^{n} p_{ih} + t_h - s_h + \Delta + p_{\pi_j A}$$

On an other hand, we have: $\min(p_{i_1 h}, p_{\pi_j A}) \le \min(p_{i_1 A}, p_{\pi_j h})$

- If $\min(p_{i_1 h}, p_{\pi_j A}) = p_{i_1 h}$ then $p_{\pi_j h} \ge p_{i_1 h} > \Delta$ so

$$C_{\max}(\pi) \le \sum_{1\le i\le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_j A} \le (\sum_{1\le i\le n} p_{ih} + t_h - s_h) + (p_{\pi_j h} + p_{\pi_j A}) \le 2C^*_{\max}$$

- If $\min(p_{i_1 h}, p_{\pi_j A}) = p_{\pi_j A}$ then $p_{\pi_j A} \le p_{i_1 A}$ so

$$C_{\max}(\pi) \le \sum_{1\le i\le n} p_{ih} + t_h - s_h + \Delta + p_{\pi_j A} \le (\sum_{1\le i\le n} p_{ih} + t_h - s_h) + (p_{i_1 h} + p_{i_1 A}) \le 2C^*_{\max}$$

In conclusion $C_{\max}(\pi) \le 2C^*_{\max}$.

The following example shows that the bound is tight. Consider an instance with *m*=1, *n*=2, $p_{11} = 1$, $p_{21} = z$, $p_{1A} = 2$ and $p_{2A} = 2$. We also have s_1=z and t_1=z+1, where z is a positive integer. If we apply H_1 to the problem we get a sequence J_1, J_2 with $C_{\max}(\pi) = 2z + 3$. However, the optimal solution is to sequence J_2 followed by J_1 with $C^*_{\max} = z + 4$. Hence $C_{\max}(\pi) / C^*_{\max}$ approaches 2 as z approaches infinity. □

4. Availability constraint on machine M_A: $Am|nr-a(M_A)|C_{\max}$

In this problem, there is only one availability constraint on the machine M_A. The following result holds:

Lemma 3

For the $Am|nr-a(M_A)|C_{\max}$ problem, there exists an optimal solution which is a permutation schedule.

Proof:

Similar to the proof of lemma 2. □

Although lemma 3 provides a considerable space reduction, in this case the ratio $C_{\max}(\pi_H)/C^*_{\max}$, where π_H is a solution given by an arbitrary heuristic *H*, can be arbitrarily large. The following heuristic gives an example:

We consider the heuristic H_{Potts} proposed by [20] for the $Am\|C_{\max}$ problem. This heuristic has the tight worst-case ratio bound of $2-1/m$. In this heuristic we first apply *JA* to the $F2\|C_{\max}$ problem with the processing time of $J_i, i=1,2,...,n$, equal to $(p_{i1}+p_{i2}+...+p_{im})/m$ (in the first stage) and to p_{iA} (in the second stage). The found permutation is then applied to the $Am\|C_{\max}$ problem.

Lemma 5

If we apply H_{Potts} to sequence jobs for $Am|nr-a(M_A)|C_{\max}$ ($m\geq 2$) then $C_{\max}(\pi_{Potts})/C^*_{\max}$ can be arbitrary large, where π_{Potts} is the permutation given by the heuristic H_{Potts}.

Proof:

Consider an instance with *m*=2, *n*=2, $p_{11}=p_{21}=10$, $p_{12}=2$, $p_{22}=4$, $p_{1A}=7$ and $p_{2A}=8$. We also have $s_A=27$ and $t_A=z+27$, where *z* is a positive integer.

If we apply H_{Potts} to the problem we get a sequence J_1, J_2 with $C_{\max}(\pi_{Potts})=z+35$. However, the optimal solution is to sequence J_2 followed by J_1 with $C^*_{\max}=27$. Hence $C_{\max}(\pi_{Potts})/C^*_{\max}=(z+35)/27$ which can be arbitrarily large when *z* is very large. □

Remark 2: for the case *m*=1 (i.e. the $F2|nr-a(M_2)|C_{\max}$ problem), H_{Potts} is equivalent to *JA*, Lee [14] showed that $C_{\max}(\pi_{JA})\leq 2C^*_{\max}$ and that the bound is tight.

Remark 3: Suppose that the optimal makespan of $Am|nr-a(M_A)|C_{\max}$ ($m\geq 2$) is equal to s_A, and $t_A=s_A+z$ where *z* is a very large number. Let *H* be a polynomial heuristic to the problem, unless *H* gives an optimal solution, we will have $C_{\max}(\pi_H)>t_A>z$, and hence $C_{\max}(\pi_H)/C^*_{\max}>z/s_A$ which can be arbitrarily large. Hence it is impossible to find a polynomial heuristic algorithm with finite error bound (unless $NP=P$).

5. Conclusion

In this paper we have studied the nonresumable model of the two-stage assembly flow shop problem with availability constraint. We have provided two heuristics with tight worst-case ratio bound of 2 when the availability constraint is imposed on one of the first stage's machines. For the case where the gap is imposed on the assembly machine we have shown that no polynomial heuristic algorithm with finite error bound can be found. Future researches include extending the study to the semiresumable model and other performance measures.

References

[1] I. Adiri, E. Frosting, A.H.G. Rinnooy Kan (1991), Scheduling on a single machine with a single breakdown to minimize stochastically the number of tardy jobs, *Naval Research Logistics*, **38**, 261 – 271.

[2] R. Aggoune (2004), Minimizing the makespan for the flow shop scheduling problem with availability constraints, *European journal of Processing research* **153**, 534 – 543.

[3] H. Allaoui, A. Artiba (2006), Scheduling two-stage hybrid flow shop with availability constraints, *Computer and operations research* **33**, 1399 – 1419.

[4] J. Blazewicz, J. Breit, P. Formanowicz, W.L. Kubiak, G. Schmidtd (2001), Heuristic algorithms for the two-machine flow shop with limited machine availability, *Omega* **29**, 599 – 608.

[5] J. Breit (2004), An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint, *Information Processing lettrers* **90**, 273 – 278.

[6] W.J. Chen (2006), Minimizing total flow time in the single-machine scheduling problem with periodic maintenance, *Journal of the Operational Research Society* **57**, 410 – 415.

[7] T.C.E. Cheng, G. Wang (2000), An improved heuristic for two-machine flowshop scheduling with an availability constraint, *Operation Research Letters* **26**, 223 – 9.

[8] J.N.D. Gupta (1988), Two-stage hybrid flowshop scheduling problem, *Journal of the Operational Research Society* **39**, 359 – 364.

[9] M. Haouari, T. Daouas (1999), Optimal scheduling of the 3-machine assembly-type flow shop, *RAIRO Operations Research* **33**, 439 – 445.

[10] A.M.A. Hariri, C.N. Potts (1997), A branch and bound algorithm for the two-stage assembly scheduling problem, *European Journal of Operational Research* **103**, 547 – 556.

[11] S.M. Johnson (1954), Optimal two- and three-stage production schedules with setup times included, *Nav. Res. Log. Q.* **1**, 61 – 68.

[12] W. Kubiak, J. Blazewicz, P. Formanowicz, J. Breit, G. Schmidt (2002), Two-machine flow shop with limited machine availability, *European journal of operational research* **136**, 528 – 540.

[13] C.Y.Lee (1991), Parallel machines scheduling with non-simultaneous machine available time, *Discrete Applied Mathematics* **30**, 53 – 61.

[14] C.Y. Lee (1997), Minimizing the makespan in the two-machine flow shop scheduling problem with an availability constraint, *Operations Research Letters* **20**, 129 – 139.

[15] C.Y. Lee (1999), Two-machine flowshop scheduling with availability constraints, *European journal of operation research* **114**, 420 – 429.

[16] C.Y. Lee, T.C.E. Cheng, B.M.T. Lin (1993), Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem, *Management Science* **39**, 616 – 625.

[17] C.Y. Lee, S.D. Liman (1992), Single machine flow-time scheduling with scheduled maintenance, *Acta Informatica* **29**, 375 – 382.

[18] C.J. Liao, W.J. Chen (2003), Single-machine scheduling with periodic maintenance and nonresumable jobs, *Computers & Operations Research* **30**, 1335 – 1347.

[19] C.J. Liao, D.L. Shyur, C.H. Lin (2005), Makespan minimization for two parallel machines with an availability constraint, *European journal of processing Research* **160**, 445 – 456.

[20] C.N. Potts, S.V. Sevast'janov, V.A. Strusevich, L.N. Van wassenhove, C.M. Zwaneveld (1995), The two-stage assembly scheduling problem: complexity and approximation, *Operations Research* **43**, 346 – 355.

[21] C. Sadfi, B. Penz, C. Rapine, J. Blazewicz, P. Formanwicz (2005), An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints, *European Journal of Operational Research* **161**, 3 – 10.

[22] S. Xi, M. Kazuko, N. Hiroyuki (2003), Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling, *European Journal of Operational Research* **146**, 498 – 516.

Distributed Decision Making in Hospital Wide Nurse Rostering Problems

Stefaan Haspeslagh, Patrick De Causmaecker
Katholieke Universiteit Leuven Campus Kortrijk, E. Sabbelaan, 8500 Kortrijk, Belgium,
{stefaan.haspeslagh,Patrick.DeCausmaecker}@kuleuven-kortrijk.be

Greet Vanden Berghe
KaHo St.-Lieven, Information Technology,Gebr. Desmetstraat 1, 9000 Gent, Belgium,
Greet.VandenBerghe@kahosl.be

This contribution considers the distributed version of the Nurse Rostering Problem. A hospital consists of several wards that have a high degree of autonomy. Each ward maintains its local roster. This local decision making problem is of combinatorial complexity. By sharing and exchanging resources, the wards want to improve their rosters and tackle staff shortages. In this paper, a negotiation based model is studied in which hospital wards are represented by agents. The agents negotiate about their rosters: in case of staff shortage, requests are raised by the agents. An experimental setup is provided as proof of concept.

Keywords: Real World Scheduling, Rostering, Agent Based Scheduling.

1 Introduction

The purpose of this paper is to study the possibilities of a distributed approach in the case of the nurse rostering problem. We envisage a system consisting of units (wards) with a large degree of autonomy and highly detailed local decision making. The independence of these units is limited by the sharing of resources and by the opportunities or obligations for inter unit assistance. This assistance may increase the efficiency of one unit while only slightly hampering the performance of another one. On the long term, the performance of the larger systems should benefit. So, in its simplest form, we must consider two levels of decision making which we label local and global. The local system takes the highest level of detail into account. We are especially interested in the case where the local decision problem is of a combinatorial complexity. Various local details are aggregated and translated into variables that can be handled at the global level, which, at the same time, introduces supplementary requirements, constraints and preferences. The decision problem at this global level may be of a combinatorial complexity as well.

A system consisting of autonomous units naturally leads to an agent based model. Such models have been intensively studied over the last decade, e.g. by Valckenaers et al. [11]. Rather recently the link with optimisation has obtained some attention. Shen et al. [8] give an updated review in the field of intelligent manufacturing. We think that the optimising behaviour of an agent based system should be studied as close to the real world problems as possible. In this paper we study the behavior of a system of negotiating agents for a distributed nurse rostering problem. We select a specific negotiation protocol for the global problem and an algorithm for the local problem. Our aim is to demonstrate the feasibility of such an approach and to identify some relevant questions for further research. Our ultimate aim is to demonstrate the feasibility of such an approach. In this paper, we discuss requirements, propose a model and identify some relevant questions for further research.

The Nurse Rostering Problem (NRP) has been the subject of intensive study. An overview is presented by Burke et al. [4]. Typically a hospital is divided into several wards, and rosters

are generated at ward level. Rosters have to satisfy each nurse's work regulation while realizing a given coverage. Nurse preferences must be satisfied if possible. The criteria and objectives used to evaluate rosters at ward level are different from those at hospital level.

At the level of an individual ward, the criteria are those that have been analysed in previous work [4]. Demands stem from the workload per time unit or shift in the ward and are typically expressed as a number of nurses of a specific qualification. Work regulations define, sometimes nurse-specific, values for e.g. the maximum number of consecutive night shifts, the minimum and maximum number of consecutive days off, and so on. In addition to those constraints, one wants to take personal preferences and ad hoc requests of nurses into account.

At the hospital level other criteria are to be considered. Here, the manager wants to guard the fairness of the work load distribution, wants to raise certain quality levels, wants to minimize the personnel cost, wants to decide where resource shortages are allowable at peak moments ...

As an example of a local problem that may be resolved at the global level, we will consider the cases of sudden staff shortage due to absence or unexpected overload. These may be hard to deal with at the local level where a longer term schedule should not be disturbed too drastically, while chances are higher that a peer ward accidentically has some spare capacity at or about this specific moment in time. Trivedi and Warner [10] and Gascon et al. [7] introduce a pool of float nurses. They solve shortages by using nurses from this pool. Siferd and Benton [9] tackle shortages by calling nurses from other wards and by allowing nurses to work overtime. Meisels and Kaplanksi [6] pioneered a negotiation based approach. These authors introduce three stages. In stage 1, scheduling agents generate a local solution. In stage 2, a global feasible solution is constructed. The last stage is used by the agents to improve their local schedules. Meisels and Kaplanski assume a fixed pool of shared resources. Bard and Purnomo [2] propose various ways of tackling staff shortages. Relevant for the present discussion is the one in which initially negotiation takes place with a pool of floating nurses. Consquently, nurses who are not needed by the wards, are placed on an on-call list. If none of the proposed solutions suffices the hospital can call in agency nurses.

In this contribution we will use a negotiation based model for the optimisation at the global, hospital wide, level. In case of shortage, we allow for requests to be raised by wards. Other wards of the hospital consider re-rostering in order to satisfy the demand and post offers to resolve the request. The model is based on an extension of the Contract Net Protocol.

In Section 2, the model for the nurse rostering problem at the level of a ward is presented. Section 3 presents the negotiation model. In Section 4 we elaborate on the required components and their specifications as they follow from the discussions in the previous sections. Section 5 discusses the experiments we performed as a proof of concept. Section 6 draws conclusions and describes future research.

2 The nurse rostering problem for one ward

In [4] a large number of models and approaches are listed and discussed. Most authors treat the Nurse Rostering Problem as a matching exercise between the needs of the hospital and the constraints (legal and personal) under which the ward personnel operates. We use the model of [3] where coverage demands, in the simplest case, are considered hard and personnel rules and preferences are implemented through a penalty function weighting the importance of a specific constraint. As an extension, coverage demands can be considered soft as well, if desired. This complicates the optimisation procedure but may offer a richer set of options to be negotiated among agents. Figure 1 shows a common used representation for rosters. The columns stand for

the days of the planning period. The rows consist of the employees working in the ward. Every working day consist of a number of shift types (night, morning, late, ...).

	Mo	Tu	We	Th	Fr	Sa	Su
Q1	N	N		M	M	L	
Q2	L	M	M			N	N
Q3	L	L	L	L	L		M
Q4					L	L	L
Q5	M	M	N	N		M	M

Figure 1: Example roster

3 The negotiation model

The negotiation model we use in our current approach is based on an extension of the Contract Net Protocol [1] that is called the Extended Contract Net Protocol (ECNP). It is designed for use with cooperative and self-interested agents. In this protocol, managers make calls for bids. Contractors try to formulate appropriate bids to those calls. In the ECNP, managers can concurrently call for bids and a contractor agent can concurrently manage several negotiation processes with multiple managers. The protocol is more efficient in time and is fault tolerant.

From our point of view, wards can be manager and contractor at the same time. In case of a staff shortage, a ward will raise a request. A request can either be a specific call for a nurse to work a particular shift or a request to make a (partial) roster of an employee lighter. Other wards consider re-rostering in order to answer the call. Meanwhile they raise requests for their own problems. First we explain the negotiation protocol in more detail. Second, we elaborate on the benefits of this protocol. Last, we discuss the requirements that this protocol imposes.

The parties in the protocol are the the manager and the contractor. The manager is the ward that raises requests, the contractors make offers to those requests. Negotiation is carried out in five steps:

1. an announcement is made by the manager to all the possible contractors of which it believes that they are capable to solve his problem.

2. The contractors formulate a proposal. They send a PreBid message to the manager. The manager compares the received PreBids and sends a PreAccept message to the contractor that made the best offer. The other wards receive a PreReject message.

3. A contractor that receives a PreReject may trie to make a better proposal and sends a new PreBid. If a manager receives a better offer than the best offer received so far, it sends a PreAccept message to the ward of the new best bid and a PreReject to the ward of the previous best offer.

4. After a period of time, the contractors send a DefinitiveBid to the manager.

5. The manager decides which definitive offer is best and a DefiniveAccept message is sent to the ward offering the bid. A DefinitiveReject message is sent to the others.

The advantages of this extended version of the Contract Net Protocol are fourfold:

- The protocol itself gives contractors the possibility to run several negotiation processes in parallel.
- It follows that the accumulated length of all the negotiation processes between the agents is reduced. In the original protocol, the only way for the managers to reduce the length of their negotiations, was trough managing parallel negotiations.
- By introducing the PreBidding phase, the protocol reduces the number of de-commitments between the contractors.
- It avoids penalising contractors that break their commitments.

The above motivations state the appropriateness of the protocol for this application. In order to process the requests, a simulation engine has to be designed. The use of the presented protocol imposes some demands on the simulation engine. The engine must be capable of simulating local changes to the rosters. A ward manager does not want to totally change the rosters in order to satisfy demands from other wards. Also, if a bid is rejected, a small modification of the bid could be sufficient to satisfy the request.

4 Application design, requirements and specifications

This section gives an overview of the agents and software modules used in our application. Every ward is represented by four agents. A ManagerAgent represents the role of the manager in the previously discussed ECNP. Analogously, the role of the contractor is taken by the ContractorAgent. The SimulationAgent calculates and processes bids. The ProblemIdentificationAgent is responsible for raising requests. We elaborate on those four agents in more detail below. Some software modules, e.g. a communication module, needed by the agents, are provided as well. We discuss only one software module, the SimulationEngine. Figure 2 illustrates the ward design. For each agent, we state its task, identify the other agents it cooperates with and we list the software modules used by the agent:

- ManagerAgent:

  The task of this agent is to raise requests and choose between the various reply offers made by the contractors to those requests. The agent takes several considerations into account when formulating requests. For example, it may decide to only send the request to the wards it believes are capable of producing a positive answer to its question. E.g. the Intensive Care Unit and the Coronary Care Unit may be more likely to exchange nurses because they employ similarly skilled people. Sending the same request to the geriatrics ward might be without purpose. The ManagerAgent cooperates with the ProblemIdenticationAgent to formulate questions and with the SimulationAgent to evaluate bids. The CommunicationModule is used to perform the actual communication between the wards.

- ContractorAgent:

  The task of the ContractorAgent is to calculate PreBids as an answer to calls for bids from the managers. Furthermore, if a PreReject message is received, the ContractorAgent needs to adjust its offer and formulate a new one. The Contractor makes decisions according to its current views. For example, for friendly wards, the contractor may decide to make offers

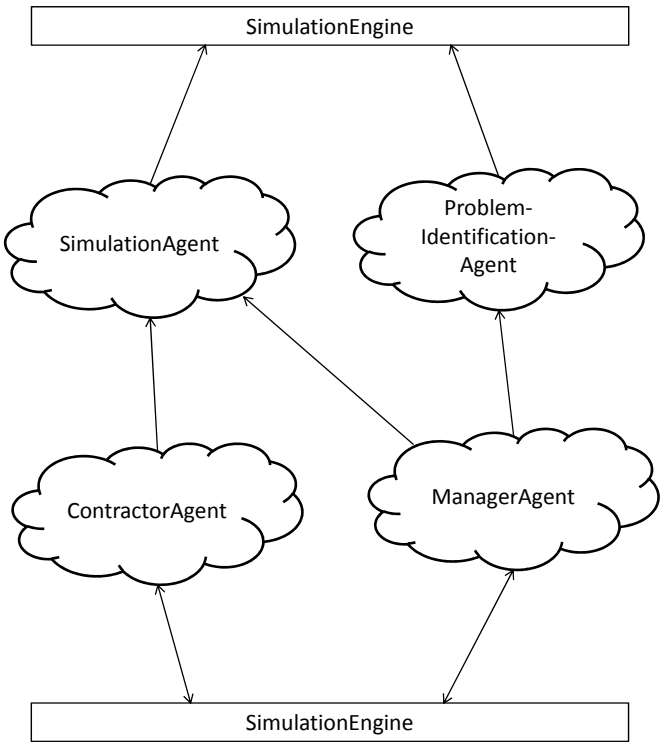


Figure 2: Ward software design

that have a more significant impact on the quality of its roster, while for other wards it wants to be more conservative. The SimulationAgent helps the contractor to calculate bids. Communication is handled by the CommunicationModule.

- SimulationAgent:

  The role of the SimulationAgent is to help the ContractorAgent with the creation of offers and to help the ManagerAgent in evaluating the bids it receives. In it simpliest form, the role of this agent could be supported by the other agents. A more advanced version should be able to learn to wich offers and questions the wards are most responsive. If the SimulationAgent detects that certain questions are never answered and that certain offers are always rejected, it should stop to raise such questions and other offers must be created. Therefore, a dedicated agent is designed. The SimulationAgent carries the beliefs and the decision making strategies of the ward manager. The SimulationEngine module performs the actual simulation. In a specific implementation, several instances of these agents and modules may be present in order to avoid bottlenecks. If this is the case, communication between the agent instances will be necessary to support the learning behaviour.

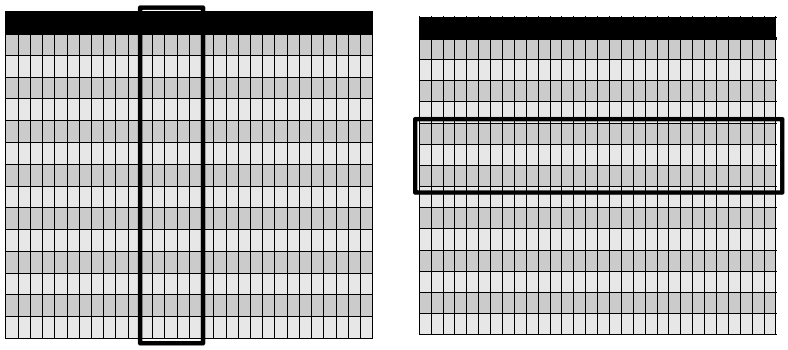
- ProblemIdentificationAgent:

  The ProblemIdentificationAgent's task is to identify problems occurring in the ward's rosters. Again, the ProblemIdentificationAgent represents the beliefs of the ward manager. Wards may differ w.r.t. their conception of when a roster is problematic. The ProblemIdentificationAgent makes use of the same SimulationEngine as the SimulationAgent, to identify problems.

The SimulationEngine allows to access the local schedule and the local planner. As stated before, the requirements of the SimulationEngine stem from the negotiation protocol used. Local changes of a ward's roster are necessary for the protocol to work. We explain what we mean by local changes. We identify three requirements:

1. The time period in which planning occurs must be adjustable. This is represented by the vertical rectangle in figure 3.1. When replanning to fit a particular shift in the roster, the ward manager does not want the entire planning period to change. Only small changes, for instance, within a time period of a couple of days before or after the time where the shift must be covered, are allowed.

2. The SimulationEngine must be capable of rerostering only for a specific number of employees. The horizontal rectangle in figure 3.2 represents this requirement. Consider for example, that a question for a head nurse is raised. In this case, the rescheduling must be restricted to the rosters of the nurses who have an equivalent skill. After all, nurses with lower skills are not qualified to cover such shifts.



(1) Vertical requirement (2) Horizontal requirement

Figure 3: SimulationEngine requirements

3. Combinations of the first two requirements. In this way, small regions in the ward's roster can be selected for rescheduling. For example, one can state that rerostering only has to be carried out for two head nurses, for one day before and one day after a particular shift. Figure 4 shows a graphical representation of this requirement.

An implementation of the SimulationEngine needs to map these requirements to the local planner's model.

5 Implementation and experimental setup

Up until now, we implemented a simplified version of the application described above. We briefly discuss the main simplifications made. The SimulationEngine does not yet comply to the requirements that we mentioned above. This is due to shortcomings in the local planner we currently use. The local planner can only reschedule an entire roster. This imposes limitations to the SimulationAgent and the ProblemIdentificionAgent. E.g., for now, the ProblemIdentificationAgent searches for the employee with the worst roster. The ManagerAgent uses that roster to raise a request. For handling requests, the SimulationAgent tries to fit every shift of the problematic roster into its ward's roster. A shift can be covered if the total penalty of the roster does not increase by some

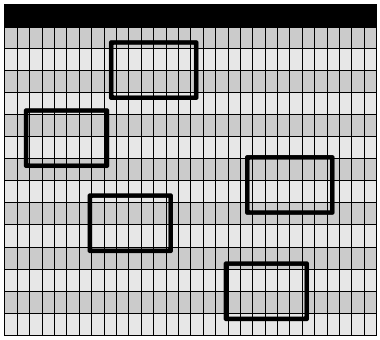


Figure 4: Combination requirement

percentage. Currently, the ManagerAgent and ContractorAgent do not build long term relations with wards. Every ward is considered equal and every request is sent to every ward.

We set up an experiment with five wards. The data for each ward's roster and the local planner is found at [5]. The wards negotiate on their rosters in order to obtain a better global solution.

6 Conclusion and future research

With the experiments described above, we want to deliver a proof of concept. However, the current experimental setup is limited. The number of possible personnel exchanges, handled by negotiation, is far less than the number of solutions considered by a local solution method. This must have its consequences for the complexity that can be handled at both levels. The present model does not yet take this into account. Limitations of the implementation are the following:

1. the agents do not keep track of the history. E.g., both the ManagerAgent and the ContractorAgent should use the information learned from previous negotiations for further negotiation. They can use that information to build a list of friendly wards. Those friendly wards can then be favoured in future negotiations. The SimulationAgent could maintain a list of possible answers to requests and could make a classification of those answers. Analogously, the ProblemIdentificationAgent can maintain a set of frequently occurring problems and the possible request associated with those problems.

2. the local planner that we currently use, is too limited in functionality. While discussing the SimulationEngine, we elaborated on the requirements in functionality of the local planning engine. Our current planner does not comply to those requirements.

3. the expressive power of the local nurse rostering model. For instance, for friendly wards, the ContractorAgent may consider offers with a significant impact on the ward's penalty. This aspect is not incorporated into the current objective function.

In future research, the influence of the negotiation protocol on the SimulationEngine's requirements, needs to be investigated. We previously identified the requirements imposed by the Extended Contract Net Protocol. Analogous work needs to be done with other negotiation models. Next, the results obtained by negotiating according to those protocols, have to be compared. Then, a comparison with results from central approaches is necessary.

References

[1] S. Aknine, S. Pinson, and M. F. Shakun (2004), An extended multi-agent negotiation protocol, *Autonomous Agents and Multi-Agent Systems* **8**(1), 5 – 45.

[2] J. Bard and H. Purnomo (2004), Real-time scheduling for nurses in response to demand fluctuations and personnel shortages, In *Proceedings of the fifth International Conference in the Practice and Theory of Automated Timetabling, Pittsburgh*, September 6-10, 67 – 87.

[3] E. K. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe (2001), A memetic approach to the nurse rostering problem. *Applied Intelligence* **15**(3), 199 – 214.

[4] E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem (2004), The State of the Art of Nurse Rostering. *Journal Of Scheduling*, **7**(6), 441 – 499.

[5] T. Curtois. Nurse rostering problems and benchmarks, `http://www.cs.nott.ac.uk/~tec/NRP/`.

[6] E. Kaplansky and A. Meisels (to appear), Distributed personnel scheduling - negotiation among scheduling agents, *Annals of Operations Research*.

[7] V. Gascon, S. Villeneuve, P. Michelon, and J. Ferland (2000), Scheduling the flying squad nurses of a hospital using a multi-objective programming model. *Annals of Operations Research* **96**(1), 149 – 166.

[8] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie (2006), Applications of agent-based systems in intelligent manufacturing: An updated review, *Advanced Engineering Informatics* **20**, 415 – 431.

[9] S. Siferd and W. Benton (1992), Workforce staffing and scheduling: Hospital nursing specific models, *European Journal of Operations Research* **60**(3), 233 – 246.

[10] V. Trivedi and M. Warner (1976), A branch and bound algorithm for optimum allocation of float nurses, *Management Science* **22**(9), 972 – 981.

[11] P. Valckenaers, K. Hadeli, B. Saint Germain, P. Verstraete, and H. Van Brussel (2006), Emergent short-term forecasting through ant colony engineering in coordination and control systems, *Advanced Engineering Informatics* **20**(3), 261 – 278.

Non-preemptive Scheduling of Distance Constrained Tasks Subject to Minimizing Processor Load

Klaus H. Ecker
Ohio University, Athens, OH, USA, ecker@ohio.edu

Alexander Hasenfuss
Clausthal University of Technology, Clausthal, Germany, hasenfuss@in.tu-clausthal.de

Scheduling problems that involve distance constrained tasks with repetitive requests are enjoying an increase in popularity in real time system design. Timing constraints defined by periods of fixed lengths for each task enforce some frequency of task execution. Alternatively deadlines could be specified relatively to the finish time of the previous execution of the same task. Hence, the scheduling algorithms deal with tasks that virtually consist of infinite chains of jobs with specified temporal distance constraints between each pair of adjacent jobs. In this paper, unlike approaches that consider a preemptive distance constrained task scheduling subject to minimize time distance between task execution, we consider a problem of non preemptive tasks systems with objective to reduce processor utilization while keeping the response times within their limits. As a consequence, free processor capacity remains to process aperiodic tasks. Moreover, as a side effect, the way the tasks get scheduled to obtain mnimal processor utilization results in jitter reduction improving system stability.

Keywords: distance constrained tasks, relative timing constraints, minimizing processor utilization

1. Introduction

In real-time systems for controlling some environment such as facilities, power plants, and as well in embedded systems for mechatronics applications, there are generally many processes that must be executed repeatedly. The specification for such processes contains several conditions including the pace at which a process has to be performed periodically, or, similarly, the requirement of a repeated processing guided by a given maximal time lag. One of the first results for scheduling preemptive periodic tasks was the fundamental paper of Liu and Layland [23]. A periodic task in this context is a computation that has to be performed repeatedly; the time axis is partitioned in equal length intervals, and in each interval one task instance is processed. Since then a lot of work has been invested to study variations of the original problem, cf. [1].
The periodic task model is the common choice in today's commercial real time systems; mostly based on fixed priority schemes they are suited for many applications. However, there are situations where absolute periods do not fully capture the nature of the environment; at that point relative schemes come into play.
There are many approaches documented, Chen et al. [3] assume periodic tasks together with the additional, as they call it, relative timing constraints, a *low and high jitter* for the distance between two consecutive jobs. Other generalizations regard processing times. Mok et al. [24] consider a multiframe model where the tasks are instantiated periodically, but with different execution times in each interval. The model discussed in [9, 5] assumes the periodic execution of a set of tasks with precedences and relative timing constraints in form of min/max conditions between start and finish times of any two instances. Furthermore, upper and lower bounds for job execution times are assumed. A similar model is *end-to-end scheduling*, as considered e.g. by Gerber et al. [13, 14, 15], which deals with the scheduling of sets of tasks with precedences, deadlines, various constraints, and communication delays. Another model modification is discussed in [6] where repeating processes are considered, and the time between the processes is determined at each iteration step by a so-called dynamic temporal controller. In [4], Choi and Agrawala distinguish between static periodic processes and processes arriving dynamically (called aperiodic or sporadic). *Aperiodic* tasks are characterized by a given arrival time, a release (or ready) time, a deadline, and a worst case

execution time. *Multiple instance tasks* which are divided into periodic tasks and sporadic tasks are repeatedly invoked. The behavior of the classical periodic model in presence of aperiodic tasks is analyzed in [2]. Aperiodic tasks without deadlines in dynamic priority systems are considered in [26]. A *sporadic task* is characterized as a sequence of task instances (also called jobs). Between two consecutive jobs executions there is a given minimum inter-arrival time δ, meaning that if an instance is invoked at some time t, the next instance is invoked at any time t' satisfying $t' \geq t + \delta$. This way it is ensured that the frequency of sporadic occurrences is bounded above. It is assumed that the sporadic tasks are included dynamically, i.e. on-line at run-time, into the schedule.

Many technical systems have distance constrained requirements and function in a temporal distance-constrained manner, rather than just periodically. Such a model is defined by Hun, Lin et. al. in [20, 17]. In this model, the temporal distance between any two adjacent instances of the same task is bounded by some given upper bound. Scheduling schemes, schedulability conditions, and fundamental properties are given. Modifications of algorithms designed for periodic tasks systems and for the pinwheel problem to schedule distance constrained tasks are presented.

Examples of special cases of relative timing constraints are task couplings as considered e.g. in [15, 16, 25], where in any feasible schedule an upper bound gap, a lower bound gap, or an exact gap between jobs of specified pairs are required. Krings et al. [22] also consider coupled jobs where an upper bound for the gap is specified.

In this paper, we study a problem for distance constrained tasks on a single processor, and concentrate on the algorithms that minimize processor utilization. We show that the processor utilization can be reduced considerably while the response times are still within their limits; hence, the processor has more free capacity to process sporadic tasks. As a side effect, the way the tasks get scheduled to obtain minimal processor utilization results in jitter reduction improving system stability.

The paper is organized as follows: The following section introduces the model, specifies notation, and formulates the problem. Next, we present closely related models and transfer complexity issues. Then, simulation results are presented showing that processor utilization in the distance constrained model tends to be smaller than in the corresponding periodic model. As a theoretical result, exploring the border to intractability, we show a polynomial time solution for the two-task problem with zero release times that minimizes processor utilization. The last section addresses the summary and outlook on further problems left open for future research or being subject of work in progress.

2. Model and Problem Formulation

We consider the problem of non-preemptively scheduling sets of tasks on a single processor, each task representing an unbounded chain of task instances (or jobs) with relative release times and deadlines.

Distance Constrained Task Model

Given a set of tasks $\mathsf{T} = \{T_1, \ldots, T_n\}$, we assume that they are independent of each other. Each task T_j $(1 \leq j \leq n)$ is characterized by a triple of parameters, (p_j, r_j, d_j): processing time p_j, release time r_j, and deadline d_j. For a given task we assume that all its instances have the same processing time and relative timing constraints. A task T_j is the representative of a job sequence of unbounded length, $T_j = (T_j,^1, T_j,^2, \ldots)$, where the jobs are chain-dependent $(T_j,^1 \prec T_j,^2 \prec \ldots)$. We use the same notation for jobs as for tasks, but enhanced by the instance number. We say that two task instances $T_j,^i$ and $T_j\ ,^{i+1}$ are coupled by relative timing constraints (relative release time and relative deadline) in the following sense: if $T_j\ ,^i$ completes at time $c_j,^i$, then the next instance, $T_j\ ,^{i+1}$, must not start before time $c_j,^i + r_j$, and complete not later than $c_j,^i + d_j$. Observe that $d_j \geq r_j + p_j$ for feasibility reasons. The values r_j and d_j are referred to as *relative release time* and *relative deadline*, respectively.

Problem Formulation

We assume that the problem of scheduling a given set of tasks is deterministic, i.e. that all timing parameters (processing times, relative release times and deadlines) are known and specified in ad-

vance, so that a schedule can be constructed off-line. The objective is to schedule the tasks non-preemptively on a single processor. Without loss of generality it is assumed that all time parameters are integers. For convenience reasons we omit the word "relative" and use the terms release time and deadline instead of relative release time and relative deadline, respectively. To define a reference point (offset) for the release times and deadlines of the first job T_j, ^{1} of each task T_j, we assume that it is set to 0.

A non-preemptive schedule for a set of tasks $T = \{T_1, \ldots, T_n\}$ on a single processor can be described by a sequence of jobs and idle intervals. For example, the sequence (T_j ,^{1}, I_k , T_j ,^{2}, I_k , ...) represents the schedule where the first instance T_j ,^{1} of task T_j is followed by an idle interval (idle "instance") of length k, then followed by the second instance T_j ,^{2}, etc. To simplify notation we just specify it by the corresponding sequence of tasks with idle intervals (T_j , I_k , T_j , I_k , ...). Since each task has an unbounded number of jobs, the schedules are of infinite length. On the other hand, if a feasible schedule for a given set of tasks exists, there will also be one with cyclic behavior that can be described by a job sequence of finite length (cf. [FC02], Th. 2.1; [10], Th. 27.1). A partial schedule is referred to as a sequence of jobs (and idle intervals between them) of finite length. The concatenation of partial schedules S_1 and S_2 is denoted by S_1S_2. If S is a partial schedule, S ,– denotes the schedule obtained by infinite repetition of S. We say that S is a generating sequence of S ,–. To check the feasibility of a schedule it is easy to see that it suffices to check the feasibility of a partial schedule. In this paper, we consider the following problem: If a feasible solution exists, find a schedule that minimizes processor utilization.

3. Related Models

In this section, we briefly describe the relation of the DC model with closely related scheduling problems.

Comparison with Periodic Task Model

If we consider distance constrained (DC) tasks in a periodic paradigm, the period must be chosen such that the time lags between succeeding instances of the task never exceed $d_j - p_j$. The period π_j in a corresponding task system then calculates to $\pi_j = (p_j + d_j)/2$. This follows from the fact that it may happen that one instance is processed at the beginning of a period, and the next one at the end of the following interval. When comparing the processor utilization by task T_j we get: in the periodic task model: $p_j/\pi_j = 2p_j/(p_j + d_j)$, and in the DC model: p_j / d_j, assuming maximum gaps $p_j - d_j$ between the instances.

Comparing the processor loads we see that in the periodic model the processor utilization is up to $2d_j/(p_j + d_j)$ times larger than in DC model, or almost a factor of 2 more in case of small processing times. The following example illustrates this fact. Table 1 specifies five tasks with processing times and distance constraints. The last line shows the periods as calculated from the equation for π_j.

	T_1	T_2	T_3	T_4	T_5
p_1	5	3	7	4	8
d_1	93	77	55	42	44
π_1	49	40	31	23	26

Table 1. *An example task set with distance constraints and corresponding periods*

In the periodic model, the processor utilization is $U^{per} := \Sigma\, p_j / \pi_j = 0.88$. Notice that neither a preemptive nor a non-preemptive periodic schedule exists. In the DC model, $\Sigma\, p_j / d_j = 0.50$ is the lower bound for the utilization U^{DC} and, hence, named also in the literature the total density of task set.

An example of a feasible non-preemptive schedule with DC is ($T_4\, T_5\, T_3\, T_2\, I_{20} T_5\, T_3\, T_2\, T_1\, I_{15}$, $\overline{\qquad\qquad\qquad\qquad}$) (executed cyclically), following notations of

the section 2, with $U^{rel} = 0.58$. It shows that replacing the DC model by a periodic model can only be done at the cost of processor utilization, or it may even lead to a non-feasible problem. On the other hand, the periodic task model can always be replaced by a relative timing model by setting $d_j := 2\pi_j - p_j$.
Lower utilization demand is an important issue in practice if sporadic tasks need to be included. It would therefore be important to have sufficiently free utilization capacity available. Simulation results are presented in Figure 1. They show mean free processor capacity of the DC model (bold line) in comparison with the periodic model (dotted line). It can be seen that the processor utilization of relative timing schedules is smaller than in a corresponding periodic model.

Relationship with Pinwheel Problems
The well known *Pinwheel Problem* (PP) [21] is closely related to the DC model, in the sense that every instance of PP can easily transformed into an instance of the DC model by setting specific deadlines, zero release times, and unit processing times. But in the PP the processor utilization is always dense. Several algorithms for scheduling of pinwheel instances are devised in [8, 7]. Associated real time problems, properties of PP and relationship with other models are discussed in [10]. The pinwheel is viewed as a special case of periodic tasks scheduling problem, the discrete version of the distance constrained tasks system problem, proportional fairness, periodic maintenance problems and others. The *Generalized Pinwheel Problem* (GPP), introduced by Feinberg and Curry [12], is based essentially on the same model as DC, so results concerning complexity etc. can be transferred. The problem does not aim on minimization of utilization, though.

Complexity Considerations
In [11] it is shown that the Generalized Pinwheel Problem is NP-hard. Since GPP is based essentially on the same model as DC and constitutes a subclass, we can conclude that the DC scheduling problem is also NP-hard.

Theorem: *The problem of finding a feasible schedule in the distance constrained task model is NP-hard.*

Scheduling parameters:
$n \in 2..10$; $p_j \in 1..15$;
$\pi_j \in 20..50$,
each chosen with even distribution.

For each instance, the corresponding relative timing constraints are calculated

Total number of instances: 18152

Simulation results are shown in Figure 3.

Number of instances for which
- no periodic non-preemptive schedule exists: 4763
- no periodic preemptive schedule exists: 169

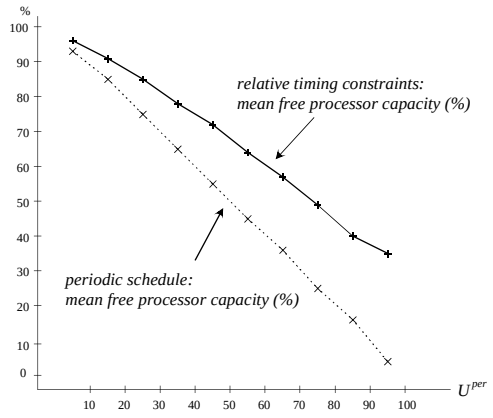


Figure 1. *Mean free processor capacity*

3. Minimizing Processor Utilization

We now turn to the optimization problem to find a schedule with minimum processor utilization. We give an answer to this question for the simple case of two tasks T_1 and T_2 with zero release times.
Notice that, as long there are no sporadic tasks, and only feasibility is concerned, there is no need to include idle intervals on the processor. The only condition for schedulability is that an instance of T_2 can be included between two instances of T_1, and vice versa. Hence, the following is trivial:

Lemma 1. *Tasks T_1 and T_2 with zero release times can be feasibly scheduled iff* $p_1 + p_2 \le min\{d_1, d_2\}$. □

In the following, we assume that there exists a feasible schedule for $\{T_1, T_2\}$. Let w.l.o.g. be $d_1 \le d_2$, and define $p = p_1 + p_2$.

Since we can restrict to cyclic schedules it is sufficient to consider partial schedules S of finite lengths, for which $S,^{\infty}$ is feasible. Let C_S be the makespan of S; if S has n_j instances of T_j (j = 1, 2), then the utilization within the time span C_S is $U = (n_1p_1 + n_2p_2)/C_S$. Even S is repeated infinitely often, the utilization of $S,^{\infty}$ is U.

A general guideline for minimizing utilization would be to try to schedule the instances of each task at maximum possible distances, i.e. $d_j - p_j$ for T_j, j = 1, 2. If the task instances can be scheduled at these distances, the utilization of S is $U = p_1/d_1 + p_2/d_2$.

Lemma 2. *The tasks T_1 and T_2 with zero release times can be scheduled at distances $d_1 - p_1$ for T_1, and $d_2 - p_2$ for T_2 without conflict if and only if $p \le gcd(d_1,d_2)$, where gcd is the greatest common divisor.*

Proof. Let S^j be a schedule for the T_j only, where the first instance of T_j starts at time 0, and the following jobs are scheduled at maximal possible distances; then at each time id_j an i-instance of T_j is started. The question is whether both schedules, S^1 and S^2, can be merged into one schedule in such a way that the distances between the jobs are kept, and no overlap occurs.

It is a well-known fact that, given two integers d_1 and d_2, we can always find integers k_1 and k_2 such that $k_2d_2 - k_1d_1 = gcd(d_1, d_2)$. It is clear that $s = gcd(d_1, d_2)$ is the smallest distance > 0 between start times of jobs in S^1 and S^2. A merged schedule, S, is constructed by scheduling the jobs of T_1 at times id_1 and those of T_2 at times $id_2 + p_1$ (i = 0, 1, ...). Let integers k_1 and k_2 be such that $k_2d_2 - k_1d_1 = s$. Since the $(k_1+1)^{st}$ job of T_1 starts at k_1d_1, the $(k_2+1)^{st}$ job of T_2 starts at time $k_1d_1+p_1$ and completes at time $k_1d_1 + p_1 + p_2 \le k_1d_1 + s = k_2d_2$. Hence the tasks can be scheduled without overlap in S if and only if $p = p_1 + p_2 < s$. □

If $p_1 + p_2 > \gcd(d_1,d_2)$, we see that the maximal possible distances cannot be kept because of conflicts. In the following we analyze this situation in detail.

We may assume that the schedule S has jobs T_1^k, T_2^l scheduled such that there is no idle interval in between. If there is no such a pair we can shift the schedule from the second job on to the left until there is no gap between them. With this assumption, it is easy to see that there are three types of cycles (types I, II, and III), as depicted in Figures 6 – 8. Types I and II differ by the task that is selected first. Type III is a combination of types I and II.

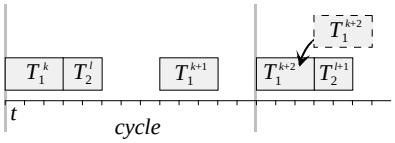


Figure 2 Cycle of type I

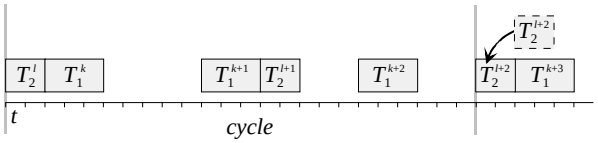


Figure 3 Cycle of type II

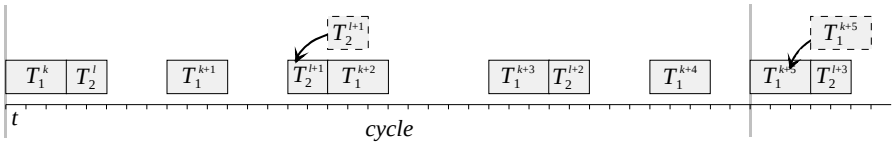


Figure 4 Cycle of type III

Let $n^{I},_1$, $n^{II},_1$, $n^{III},_1$, $n^{I},_2$, $n^{II},_2$, $n^{III},_2$ be the respective maximal numbers of T_1 and T_2 jobs in the cycles of type I, II and III at maximum possible distances as shown in Figures 6 – 8. Our aim is to determine for each cycle type the smallest values for these numbers.

Lemma 3. (i) $n^{I},_2 d_2 < n^{I},_1 d_1 < n^{I},_2 d_2 + p$;

(ii) $n^{II},_1 d_1 < n^{II},_2 d_2 < n^{II},_1 d_1 + p$.

Proof. (i) follows directly from the conditions for building type-I cycles; (ii) is symmetric.

We show this for type II cycles: let t_1 and t_3 be the respective start and completion time of the first instance of T_1 that overlaps with the instance of T_2 : $t_1 = t + n^{II},_1 d_1 + p_2$, $t_3 = t + n^{II},_1 d_1 + p$ (where t is the start time of the cycle). The overlapping starts at time $t_2 = t + n^{II},_2 d_2$ and completes at time $t_4 = t + n^{II},_2 d_2 + p_2$. Thus the condition for an overlap is: $t_3 > t_2$ and $t_1 < t_4$ (see Figure 9), i.e. $n^{II},_1 d_1 < n^{II},_2 d_2 < n^{II},_1 d_1 + p$. □

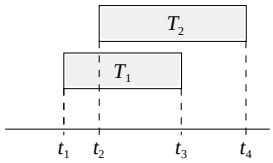


Figure 5 *Overlapping task instances*

***Cycle of type* I:** This cycle starts with an instance of T_1, followed immediately by an instance of T_2. The conflict defining the end of the cycle is resolved by moving the instance of T_1 in front of the instance of T_2 (see Figure 2). The start time of the T_1 instance then defines the end of the cycle.

Lemma 4. *$n^{I},_1$ is the smallest natural number for which $n^{I},_1 d_1/d_2 - \lfloor n^{I},_1 d_1/d_2 \rfloor < p/d_2$, and $n^{I},_2 = \lfloor n^{I},_1 d_1/d_2 \rfloor$.[1] The cycle of type* I *has length $C^{I} = n^{I},_2 d_2$, and the processor utilization is $U^{I} = (n^{I},_1 p_1 + n^{I},_2 p_2)/C^{I}$.*

Proof. Let t be the time when the cycle starts. Since T_2 keeps its maximum distances, the obtained cycle lasts from t to $t + n^{I},_2 d_2$, and we get

$C^{I} = n^{I},_2 d_2$ (1)

for the length of the type I cycle. Let $\rho = p/d_2$ and $\varepsilon = d_1/d_2$, then $n^{I},_2 < n^{I},_1 \varepsilon < n^{I},_2 + \rho$ from Lemma 3 (i).

(i). Separating the integer part of $n^{I},_1 \varepsilon$, i.e., $n^{I},_1 \varepsilon = \lfloor n^{I},_1 \varepsilon \rfloor + \alpha$, we get $n^{I},_2 < \lfloor n^{I},_1 \varepsilon \rfloor + \alpha < n^{I},_2 + \rho$, which, since $0 < \alpha, \rho < 1$, is only possible if $n^{I},_2 = \lfloor n^{I},_1 \varepsilon \rfloor$ and $\alpha = n^{I},_1 \varepsilon - \lfloor n^{I},_1 \varepsilon \rfloor < \rho$.

The minimally condition for $n^{I},_1$ leads to the result that $n^{I},_1 > 0$ is the smallest integer for which $n^{I},_1 \varepsilon - \lfloor n^{I},_1 \varepsilon \rfloor < \rho$.

The processor utilization U^{I} of this cycle is given by

$U^{I} = (n^{I},_1 p_1 + n^{I},_2 p_2)/C^{I}$. □

The other possibility of resolving the conflict, i.e., shifting the T_2-instance in front of the T_1-instance, leads to a cycle of type III. This is discussed below (see *cycle of type III*).

***Cycle of type* II:** This cycle starts with a T_2-instance, followed immediately by a T_1-instance. The conflict defining the end of the cycle is resolved by moving the T_2-instance in front of the T_1-instance (see Figure 3). The start time of the T_2-instance then defines the end of the cycle.

Lemma 5. *$n^{II},_1$ is the smallest natural number for which $n^{II},_1 d_1/d_2 - \lfloor n^{II},_1 d_1/d_2 \rfloor > 1 - p/d_2$, and $n^{II},_2 = \lfloor n^{II},_1 d_1/d_2 \rfloor + 1$. The length C^{II} of the cycle is $n^{II},_1 d_1$, and the processor utilization is $U^{II} = (n^{II},_1 p_1 + n^{II},_2 p_2)/C^{II}$.*

Proof. As compared to the cycle of type I, the roles of T_1 and T_2 are now interchanged. Hence the length C^{II} of the type-II cycle is

[1] $\lfloor a \rfloor$ denotes the largest integer $\leq a$.

$C^{II} = n^{II},_1 d_1$, (2)

and from Lemma 3 (ii) we get (ε and ρ are defined as in the proof of Lemma 4)

$n^{II},_1 \varepsilon < n^{II},_2 < n^{II},_1 \varepsilon + \rho$.

Define $\beta = n^{II},_1 \varepsilon - \lfloor n^{II},_1 \varepsilon \rfloor \in (0,1)$, then

$\lfloor n^{II},_1 \varepsilon \rfloor < n^{II},_2 - \beta < \lfloor n^{II},_1 \varepsilon \rfloor + \rho = \lfloor n^{II},_1 \varepsilon \rfloor + 1 - (1 - \rho)$,

which is only possible if $\beta = n^{II},_1 \varepsilon - \lfloor n^{II},_1 \varepsilon \rfloor > 1 - \rho$ and $n^{II},_2 = \lfloor n^{II},_1 \varepsilon \rfloor + 1$. Hence from the minimally condition we get that $n^{II},_1$ is the smallest natural number for which $\beta = n^{II},_1 \varepsilon - \lfloor n^{II},_1 \varepsilon \rfloor > 1 - \rho$.

The processor utilization U^{II} of this cycle is given by

$U^{II} = (n^{II},_1 p_1 + n^{II},_2 p_2)/C^{II}$. □

The other possibility of resolving the conflict, i.e., shifting the T_1-instance in front of the T_2-instance, is discussed next.

***Cycle of type* III:** A cycle of type III is constructed by combining cycles of type I and II (see Figure 4): Starting with a cycle of type I, we shift the conflicting T_2-instance in front of the T_1-instance, which results in the initial condition of the type II cycle. The conflict at the end of the cycle is resolved by shifting the T_1-instance in front of the T_2-instance. This leads to the initial configuration of the type-I cycle. The cycle length C^{III} can then easily be determined as

$C^{III} = n^{I},_1 d_1 + n^{II},_2 d_2 - p$

Notice that the number of T_1- and T_2-instances in this cycle is $n^{III},_1 = n^{I},_1 + n^{II},_1$, $n^{III},_2 = n^{I},_2 + n^{II},_2$, respectively. Hence the processor utilization is $U^{III} = [(n^{I},_1 + n^{II},_1)p_1 + (n^{I},_2 + n^{II},_2)p_2] / (n^{I},_1 d_1 + n^{II},_2 d_2 - p)$.

Comparing utilizations: It turns out that, for particular pairs of tasks, any of the cycles may have minimum processor utilization. However, it can be shown that the cases I or II outperform case III if U^{I} and U^{II} are not too close.

Theorem 1. *Given tasks T_1 and T_2 with zero release times, and $p_1 + p_2 > gcd(d_1, d_2)$. If T_1 and T_2 can be feasibly scheduled then the schedules S^{I}, S^{II}, and S^{III} obtained by executing the respective cycles of type* I, II, III *are feasible. The optimal schedule can be found among these schedules, and the respective processor utilizations are*

$$U^{I} = (n^{I},_1 p_1 + n^{I},_2 p_2)/C^{I}$$
$$U^{II} = (n^{II},_1 p_1 + n^{II},_2 p_2)/C^{II} \quad (3)$$
$$U^{III} = [(n^{I},_1 + n^{II},_1)p_1 + (n^{I},_2 + n^{II},_2)p_2]/C^{III} .$$

Furthermore, (*i*) *if $U^{II} \geq U^{I}(1 + p/C^{II})$ then $U^{I} < U^{III} < U^{II}$,* and (*ii*) *if $U^{I} \geq U^{II}(1 + p/C^{I})$ then $U^{II} < U^{III} < U^{I}$.*

Proof. Schedule S^{I} (and similarly S^{II} and S^{III}) is obtained by repeated execution of the cycle of type I. The feasibility of S^{I}, S^{II} and S^{III} follows from the condition $p_1 + p_2 \leq \min\{d_1, d_2\}$ and the above discussion of the cycle types.

From Lemma 3, and equ. (1), (2) we get $C^{I} + C^{II} - p < C^{III} < C^{I} + C^{II} + p$. With this, (*i*) and (*ii*) follow directly from (3). □

4. Summary and Outlook

In this paper we presented results of our ongoing work on a topic in hard real time systems. We analyzed the problem of scheduling unbounded sequences of tasks with relative timing constraints subject to minimizing processor utilization. The presented problem turned out to be NP-hard in general, but we gave an in-depth analysis of the two-task case that was shown to be solvable in polynomial time.

Many problems remain to be worked on in future research.

The introduced distance constrained (DC) model comprises the classical periodic model in the sense that the periodic parameters can be transformed to distance constrained conditions. More-

over, the mean processor utilization tends to decrease while changing to the DC model. Most commercial real time system kernels, though, are based on priority schemes. Usually they feature only few fixed priority levels, an implementation of the DC model on top of those kernels is neither easy nor efficient, cf. [1]. But integrating the DC model into a real time system kernel is subject of ongoing development.

As a consequence of minimizing processor utilization, free processor capacity remains to process aperiodic tasks. But at arrival of aperiodic tasks, the system has to decide what to do depending on a schedulability analysis. Different mechanisms, algorithms and complexity issues concerning this schedulability analysis are also of interest in further research. Moreover, what is the maximal allowed size and frequency of a sporadic task to be accepted?

Since there is no necessary and sufficient condition that can be checked in reasonable time, unless P=NP, the question about heuristic approaches and polynomial solvable subclasses is obvious. Also the exploration of the border to intractability is of interest. E.g. what is the time complexity of the three-task case? Is there still a polynomial solution? Some of these topics are already answered or dealt with, respectively, in a comprehensive theoretical framework currently developed.

Finally, we summarize some of the advantages of the relative timing approach as compared to the periodic approach:

(a) Periodic requirements can always be replaced by relative timing constraints; in this sense the relative timing approach is more general.

(b) Generally, a relative timing schedule allows including longer sporadic tasks than a corresponding periodic schedule.

(c) If an off-line schedule is required, the periodic model may lead to very long schedules where the schedule length is defined by the least common multiple of the period lengths. In contrast, the relative timing model leads in average to considerably shorter schedules.

References

[1] G. Buttazzo (2005), Rate Monotonic vs. EDF: Judgment Day, *Real-Time Systems* **29**(1), 5 – 26.

[2] E. Bini, G. Buttazzo (2003), Rate Monotonic Analysis: The Hyperbolic Bound, *IEEE Transactions on Computers* **52**(7), 933 – 942.

[3] S.-T. Cheng, A. K. Agrawala (1995), Allocation and scheduling of real-time periodic tasks with relative timing constraints, *2nd International Workshop on Real-Time Computing Systems and Applications*, 210 – 217.

[4] S. Choi, A. K. Agrawala (1997), Scheduling aperiodic and sporadic tasks in hard real-time systems, Report CS-TR-3794, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland.

[5] S. Choi, A. K. Agrawala (2000), Dynamic dispatching of cyclic real-time tasks with relative timing constraints, *Real-Time Systems* **19**(1), 5 – 40.

[6] S. Choi, A. K. Agrawala, L. Shi (1997), Designing dynamic temporal controls for critical systems, Report CS-TR-3804, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland.

[7] M.Y. Chan, F. Chin (1992), General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction, *IEEE Transactions on Computers* **41**(6), 755 – 768.

[8] M.Y. Chan, F. Chin (1993), Schedulers for Larger Classes of Pinwheel Instances, *Algorithmica* **9**, 425 – 462.

[9] S. Choi (1997), Dynamic Time-Based Scheduling for Hard Real-Time Systems. Ph.D. Thesis, University of Maryland.

[10] D. Chen, A. Mok. (2004), The Pinwheel: A Real-Time Scheduling Problem. In Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRT Press, chapter 27, 1 –17.

[11] E.A. Feinberg, M.A. Bender, M.T. Curry, D. Huang, T. Koutsoudis, J.L. Bernstein (2002), Sensor resource management for an airborne early warning radar, *Proceedings of SPIE*, Signal and Data Processing of Small Targets, Vol. 4728, 145 – 156, August 2002.

[12] E.A. Feinberg, M.T. Curry (2005), Generalized Pinwheel Problem, *Mathematical Methods of Operations Research* **62**, 99 – 122.

[13] R. Gerber (1995), Guaranteeing end-to-end timing processes', Proc. IEEE Real-Time Systems, 192 – 203.
[14] R. Gerber, S. Hong, M. Saksena (1995), Guaranteeing real-time requirements with resource-based calibration of periodic processes', *IEEE Transactions on Software Engineering* **21**(7), 579 – 592.
[15] R. Gerber, W. Pugh, M. Saksena (1995), Parametric dispatching of hard real-time jobs, *IEEE Transactions on Computers* **44**(3).
[16] J.N.D. Gupta (1996), Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per task and time lags, *Journal of Global Optimization* **9**, 239 – 253.
[17] C.-W. Hsueh and K.-J. Lin (2001), Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model, *IEEE Transactions on Computers* **50**, 51 – 66.
[18] C.-C. Han, K.J. Lin (1989), Scheduling jobs with Temporal Consistency Constraints, Proc. Sixth IEEE Workshop Real Time Operating Systems and Software, 18 – 23, Pittsburgh.
[19] C.-C. Han, K.J. Lin (1992), Scheduling distance-constrained real-time tasks, IEEE Real-Time Systems Symposium, 300 – 308.
[20] C.-C. Han, K.J. Lin, C.-J. Hou (1996), Distance-constrained Scheduling and Its Application to Real-Time Systems, *IEEE Transactions on Computers* **45**, 814 – 826.
[21] R. Holte, L. Rosier, I. Tulchinsky, D. Varvel (1992), Pinwheel scheduling with two distinct numbers, *Theoretical Computer Science* **100**, 105 – 135.
[22] A. W. Krings, M. H. Azadmanesh (1999), Avoiding run-time infeasibility in systems containing coupled tasks, *INFOR (Information Systems & Operational Research) Journal* **37**(1), 77 – 88.
[23] C.L. Liu, J. W, Layland (1973), Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM* **20**(1), 46 – 61.
[24] A.K. Mok, D. Chen (1997), A multiframe model for real-time tasks, *IEEE Transaction on Software Engineering* **23**(10), 635 – 645.
[25] A.J. Orman, C.N. Potts (1997), On the complexity of coupled-job scheduling, *Discrete Applied Mathematics* **72**, 141 – 154.
[26] M. Spuri, G. Buttazzo (1996), Scheduling Aperiodic Tasks in Dynamic Priority Systems, *The Journal of Real-Time Systems* **10**(2), 179 – 210.

Medium-Term Production and Staff Planning in the Automotive Industry

Claas Hemig, Jürgen Zimmermann
Clausthal University of Technology, Departement for Business Administration, Julius-Albert-Str. 2, D-38678 Clausthal-Zellerfeld, Germany, {claas.hemig, juergen.zimmermann}@tu-clausthal.de

In this paper we present a Dynamic Programming approach for an integrated production and staff planning problem in the automotive industry. The planning horizon has a length of three to six years. We focus on one shop with parallel production lines and search for a least cost solution subject to the given flexibility of the production resources. The flexibility is induced by the adjustment screws like production speed and time, permanent and temporary workers, and the distribution of a prescribed production program between the lines. In addition, we consider labor legislation, technical restrictions and in-plant-agreements. To obtain a tractable problem size, i.e. to guarantee the operational usability of the software tool to be developed, we modify the basic approach and introduce a heuristic to solve the embedded staff planning problem. A prototypical implementation shows that the approach works well.

Keywords: Production Scheduling, Real World Scheduling

1 Introduction

Since the automotive market becomes tighter and more dynamic the mid- to long-term planning of a flexible automotive production system is a task of increasing importance. Due to the various influences on its production systems the automotive industry has recognized that the mid-term planning of its production capacities and staff size obtains better solutions incorporating OR-methods. The highly complex problem arising from the concurrence of technical restrictions, labor legislation and in-company-agreements requires efficient solution techniques. In cooperation with one of our industrial partners we develop a software tool to support the production planners in several plants. After a detailed problem formulation in Chapter 2, we present a basic solution approach in Chapter 3, focussing on two subproblems to be solved. The paper is closed by a conclusion and an outlook on further research.

2 Problem Formulation

A typical automotive plant mainly consists of a body shop, a paint shop, and a final assembly. In each shop there may exist several parallel production lines that are organized as a continuous flow production. On the lines M types of products, called models, are manufactured. Each line can either be a solitary line (i.e. only one model can be produced) or a mix line (i.e. several different models can be produced). Buffers of limited capacities between the shops can store intermediate products to decouple subsequent shops. In this paper we focus on the final assembly with L parallel production lines where some models can be produced by more than one line. For simplicity we assume that at most two products can be manufactured per line, but we can easily extend the concept to more than two products per line. The following sections describe the decisions that have to be made in each period (usually a week or a month) of the planning horizon T using the

flexibility instruments of a shop. Furthermore, we describe the most important restrictions as well as the objective function of our mathematical model.

2.1 Shift Model and Cycle Time

The fundamental decisions we make for every line l and period t are the shift model $sm_{lt} \in \{1, \ldots, |\mathcal{SM}|\}$ and the cycle time $ct_{lt} \in \{1, \ldots, |\mathcal{CT}_l|\}$. Every shift model induces the number of shifts SQD taking place on one day (early, late and night shift), where $SQD = 1$ indicates only early shifts, $SQD = 2$ indicates early and late shifts etc. In addition, every shift model induces the operating time OT along the way products are manufactured as well as the working time WT as the sum of OT and the paid breaks. Both times are measured in minutes. Third, if a shift model provides work breaks shorter than labor law allows, an extra percentage of employees SM_X is needed so that the workers can alternate while taking their breaks according to law. The cycle time ct_{lt} is measured in products per minute and determines the basic demand for employees BMR during one shift.

Let us assume a shift model with an operating time of 6220 minutes in three shifts and an extra amount of employees of 5% as well as a cycle time with a value of 0.4 and a BMR of 600 employees. Then the overall demand for employees results in $BMR \cdot SQD \cdot (1 + SM_X) = 1{,}890$ employees. Independent of the shift model and cycle time we need extra employees taking into account that a certain percentage of the workers are absent due to illness or holidays. The production capacity of the considered production line is given by $OT \cdot ct = 2{,}488$ products.

The selection of the shift model and the cycle time on each line is independent from each other. As a matter of principle we can choose them arbitrarily every period, but a change of the shift model as well as the cycle time evokes significant organizational changes. Consequently, a change is allowed to occur only once in a given number of subsequent periods $SM^{min}_{sm,l}$ or $CT^{min}_{ct,l}$, e.g.:

$$ct_{lt} \neq ct_{l,t+1} \Rightarrow ct_{l,t+i} = ct_{l,t+i+1} \quad \forall\, i = 1, \ldots, \min\{CT^{min}_{ct_{l,t+1},l}, T - t - 1\} \tag{1}$$

A change of the shift model e.g. from a 2-shift- to a 3-shift-model resizes the production capacity significantly. Therefore, with respect to the demand for products it is possible to adjust the capacity by cancelling single shifts, e.g. in a 3-shift-model the night shift on Friday. This is not considered as a change of the shift model.

After each change of the cycle time the stations on a line are consolidated and/or new stations are built. In this case, the workers often are assigned to different tasks and require a learning phase ([6]). They are less productive and have to be supported by an additional fraction of workers ct^X_{lt} which is modeled by a piecewise constant function as shown in Figure 1. We divide the learning phase into three phases: In the first phase after a change of the cycle time in period t^* with a length of $CT^{LT1}_{ct,l}$ periods the value ct^X_{lt} is set to $CT^{X1}_{ct,l}$ and is greater than $CT^{X2}_{ct,l}$ in the second phase with length $CT^{LT2}_{ct,l}$. In the third phase no extra employees are needed any longer until the next change of the cycle time occurs.

2.2 Staff

The demand for employees at line l in period t, which is solely calculated from the selected shift model and cycle time of the line, can be met by permanent and temporary staff ps_{lt} and ts_{lt}, respectively. Typically, permanent employees can only be dismissed at a few points of time (e.g. at the end of a quarter) while contracts with temporary ones can expire at the end of any period. Permanent employees are trained better than their temporary colleagues and are evidently necessary

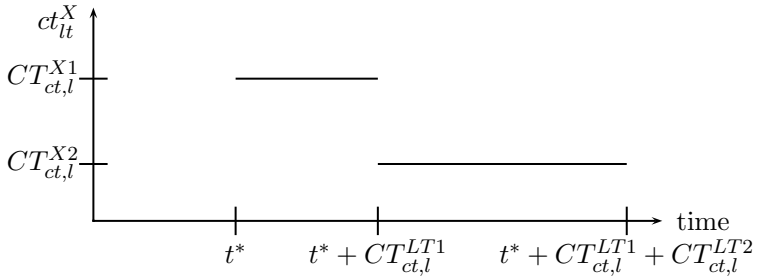


Figure 1: Extra amount of employees after changing the cycle time

for a continuously high quality. Thus, the fraction of temporary employees in the overall staff os_{lt} must not exceed a certain level TS^{max}:

$$ts_{lt} \quad \leq \quad os_{lt} \cdot TS^{max} \quad \forall\, l = 0, \ldots, L-1 \quad \forall\, t = 0, \ldots, T-1 \tag{2}$$

Especially regarding the staff the production lines cannot be decoupled from each other. For instance, the sum of all employees hired in one period t must not exceed the training capacities of the shop or due to their broad training permanent workers can be displaced them from one line to another (ds_{lkt}) to meet staff demand peaks.

2.3 Working Hours Summary

The working hours summary of a line where the workers collect their over- and undertime of each period is an important instrument to introduce flexible working hours (e.g. [3]). The over- and undertime is the difference between the attendance time WT of the selected shift model and the contracted working hours of the labor agreement. Due to the mid- to long-term planning horizon, we do not hold the summary for every single worker but for every single line as the average of all workers currently employed there, where the value whs_{lt} is bounded by in-company-agreements. Moreover, the summary has to take the value of 0 during a certain space of time (e.g. one year) or at a certain point of time (e.g. every last period of a year).

2.4 Production Program

Every period t the final assembly has to produce a certain amount D_{mt} of model m allowing a given relative deviation D^V, where the decision how many products of model m are produced on line l in period t is denoted by the variable p_{mlt}:

$$D_{mt} \cdot (1 - D^V) \leq \textstyle\sum_{l=0}^{L-1} p_{mlt} \leq D_{mt} \cdot (1 + D^V) \quad \forall\, m = 0, \ldots, M-1 \quad \forall\, t = 0, \ldots, T-1 \tag{3}$$

During a certain range of time (e.g. one year) the overall demand of every model must be satisfied.

Since the production cost for a model m on a line l only depends on the selection of the cycle time and not on the load of a production line, some simplifying implications can be inferred. First, the lines should use the available capacity as long as there is enough demand for products. Second, if the capacity exceeds the demand one or more shifts have to be canceled so that the resulting capacity can be used to the maximum. Furthermore, if two products are produced on one line, for every cycle time—with respect to in-company-agreements—a preferred mix ratio of the products is given so that all stations of the line are nearly equally loaded (e.g. [4]). From this ratio the distributed production program can deviate only in a very tight range so we can determine the main part of the program's distribution.

2.5 Objective Function

The objective is to minimize a cost function which is given as the discounted sum of the staff, production, and changing costs over all periods. The staff costs C_t^S are composed of the basic wages, shift premiums, and costs for organizational modifications deriving from their hiring, dismissal and displacement to other lines. Dismissed employees get a compensation and possibly a payoff for their overtime hours on their current working hours summary. The production costs C_t^P are the sum of the variable costs for producing model m on line l using cycle time ct. The changing costs are invoked by changing the shift model (C_{lt}^{sm}) or the cycle time (C_{lt}^{ct}) on a line.

3 A Dynamic Programming Solution Approach

3.1 Basic Approach

We can model the considered problem as a MIP with a quadratic objective function and some restrictions neither linear nor convex. Particularly, due to the calculation of the staff demand (see Section 2.2) and the working hours summary, we are confronted with a non-convex MINLP. For this reason we selected a Dynamic Programming approach (see e.g. [1], [2]) to solve the problem. The main elements of any Dynamic Programming approach are states, decisions, cost functions to evaluate a state, and transformation functions. These elements are introduced in the following.

A state z_t associated with period t is characterized by the over- or underproduction Δp_{mt} of the different models m accumulated over all previous periods as well as a number of line-specific state variables. These are the actual shift model sm_{lt} and cycle time ct_{lt}, the number of periods gone by since their last change (s_{lt}^{sm} and s_{lt}^{ct}, respectively), the number of permanent and temporary employees ps_{lt} and ts_{lt}, as well as their working hours summary whs_{lt}.

A decision x_t contains the choice of the shift model $\widehat{sm}_{lt}$ and the cycle time $\widehat{ct}_{lt}$, the number of hirings, dismissals and displacements of permanent and temporary employees Δps_{lt}, Δts_{lt}, and ds_{lkt}, and the decision about the number of products manufactured on a certain line p_{mlt}.

Given some state z_t and an associated decision $x_t(z_t)$, the transformation function $t_t(x_t(z_t), z_t)$ generates a new state z_{t+1} in the next period. The chosen shift model and cycle time is adopted by $sm_{l,t+1} = \widehat{sm}_{lt}$ and $ct_{l,t+1} = \widehat{ct}_{lt}$, respectively. The number of employees is modified as

$$ps_{l,t+1} = ps_{lt} + \Delta ps_{lt} + \textstyle\sum_{k=0}^{L-1} (ds_{klt} - ds_{lkt}) \quad \forall\, l = 0, \ldots, L-1 \quad \forall\, t = 0, \ldots, T-1, \tag{4}$$

$$ts_{l,t+1} = ts_{lt} + \Delta ts_{lt} \quad \forall\, l = 0, \ldots, L-1 \quad \forall\, t = 0, \ldots, T-1, \tag{5}$$

and the accumulated over- or underproduction is modified according to the production and the demand for each product

$$\Delta p_{m,t+1} = \Delta p_{mt} + \textstyle\sum_{l=0}^{L-1} p_{mlt} - D_{mt} \quad \forall\, m = 0, \ldots, M-1 \quad \forall\, t = 0, \ldots, T-1. \tag{6}$$

Finally, the working hours summary is calculated and (in analogy for $s_{l,t+1}^{ct}$) $s_{l,t+1}^{sm}$ is set to $s_{lt}^{sm} + 1$ if no change of the shift model took place, or set to 0 otherwise.

The costs for changing the cycle time C_{lt}^{ct} is a linear function in ps_{lt} and ts_{lt} for all $l = 0, \ldots, L-1$. The binary variables sm_{lt}^1 and ct_{lt}^1 indicate whether the correspondent changing costs have to be considered and the Bellman equation for the forward recursion can be written as

$$F_{t+1}^*(z_{t+1}) \;=\; \min \left(\left(C_t^P + C_t^S + \sum_{l=0}^{L-1} \left(ct_{lt}^1 \cdot C_{lt}^{ct} + sm_{lt}^1 \cdot C_{lt}^{sm} \right) \right) \cdot e^{-\alpha t} + F_t^*(z_t) \right). \tag{7}$$

Decisions as well as states can be infeasible and will then be eliminated. A decision is infeasible if e.g. the sum of hirings on all lines exceeds the training capacities of the shop. A state is infeasible if for instance the working hours summary is outside its admissible bounds. Infeasible decisions and those which lead to infeasible states are not generated to save computing time and memory.

The cost function serves to evaluate a state. If—disregarding costs—two states are identical then the more expensive one and the decision leading to it are eliminated. This way every state has a unique predecessor. Nevertheless, every state can have several successors that can be reached by different decisions. Starting at the least cost state z_T at the end of our planning horizon, we determine the optimal policy by backtracking through the predecessor states.

Due to the mid- to long-term planning horizon and the huge size of the feasible decision space it is reasonable and necessary to merge similar decisions to one. To give an example, we consider only decisions that—ceteris paribus—differ in the number of hired flexible workers in discrete steps of 20. Using this idea we can assure the operative usability of the developed software tool for the case of one single production line. In the case of parallel lines this simplification does not suffice to control the increasing complexity. For this reason, we adopt an idea by Schneeweiß ([5]) and separate the approach into two levels.

3.2 Separation into Top and Bottom Level

Both the distribution of the production program between the lines as well as the staff demand on the lines are dependent on the chosen shift models and cycle times. Both are independent from each other, so we can separate our approach from Section 3.1 into two levels. On the top level we generate, emanating from a feasible state in some period t, all feasible combinations of shift models and cycle times over all lines. We duplicate each of these combinations into as many decisions as there are optimal solutions with respect to the distribution of the production program between the lines (see Section 3.3). We complete each of these decisions by a greedy heuristic sizing the hiring, dismissal and displacement of staff (see Section 3.4). The quality of the heuristic for the decision concerning the staff is of utmost importance and has to take future periods into account. At this point, the decision on the top level is complete and the Dynamic Programming approach can be executed for the entire planning horizon. By this approach the number of decisions (and therefore the number of resulting states in the next period) is significantly reduced and our prototypical implementation shows the usability even for the case of parallel lines. We obtain an optimal policy over all periods keeping in mind that we use a heuristic for the staff decision.

In the second step, on the bottom level, for every period the decisions are partly preset, namely by adopting the values of $\widehat{sm}_{lt}$, $\widehat{ct}_{lt}$ and p_{mlt} from the optimal policy of the top level. We duplicate these decisions into as many as we can generate by enumeration of all meaningful decisions concerning the staff. There we rule out that e.g. permanent staff is displaced from line 1 to line 2 and simultaneously permanent staff is displaced from line 2 to line 3. Excluding such a decision is justified by the assumption that the costs for displacing staff only occurs on the line receiving the employees so that the triangle inequality always holds. Analogously, dismissing of permanent employees and simultaneous reception of permanent ones by displacements or hiring is an infeasible decision. Altogether, we complete the decisions adopted from the top level by those on the bottom level, start over the Dynamic Programming algorithm and determine an optimal policy.

3.3 Distribution of the Workload between the Production Lines

The distribution of the production program is restricted by two essentials: Firstly, by the choice of the shift model and the cycle time on every line the capacity is given, and secondly, the demand of each product and each period is given with respect to a certain relative deviation. This means

that there is a minimal demand that has to be met and an additional, optional demand that may be met up to the maximum demand.

Before starting with the distribution of the production program, we test whether the maximal capacities for every single product exceed the maximal demands and whether the total capacity exceeds the total maximal demand. In this case, we reduce the capacity by canceling single shifts. Hence, the capacities never exceed the demands. If the capacities do not suffice for the minimal demand the decision is infeasible. Now the capacity of every line producing only one model is assigned completely to the production program of the related model. For every other line we assign, as indicated at the end of Section 2.4, the main part of the capacity to the production program of the two related models. The distribution of the production program still pending is now obtained by modeling the problem as a classical transportation problem where each line asks for and each model supplies production program. The transportation unit costs are derived from the variable production costs. In particular, we separate each model, e.g. denoted with A, into two suppliers, namely A_{min} offering the remaining minimal demand to be distributed after the considerations made above, and A_{add} offering the additional demand of A. Since the additional demand has not to be produced entirely, we introduce a dummy line to balance the transportation problem. Similarly to the splitting of the products, we separate every production line into as many destinations as products can be produced on that line. In doing so, we model the preferred mix ratio of the selected cycle time on this line and its allowed deviation as mentioned before.

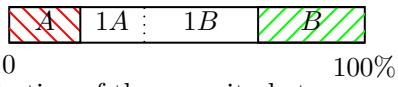


Figure 2: Distribution of the capacity between model A and B

Figure 2 illustrates the capacity of line 1 on which the models A and B can be produced. The dotted line denotes the preferred mix ratio, the left and right hatched rectangles represent the assignment of A's and B's production program to 1's capacity. A solution of the transportation problem provides the information whether the remaining, unmarked area (the yet unused capacity) should be filled with production workload of model A or B. The labels (see Figure 2) indicate the preferred model to be produced there to achieve the preferred mix ratio. This preferences are modeled by the transportation unit costs, i.e. the higher the cost the less is the preference to produce the correspondent model on the respective lines. The unit cost for the transportation of a model to the destinations built out of the line is set to the related variable production cost. The maximum of all these values is added to the unit costs for the transportation to the less preferred destinations, e.g. model A to destination $1B$. If a line cannot produce a model the transportation unit cost is set to a sufficiently large value $Big - M$. Setting the transportation unit costs, we do not differentiate between the two suppliers A_{min} and A_{add}. The costs for transporting from A_{min} to the dummy line are set to $Big - M$, from A_{min} it is set to a greater value than all transportation unit cost to any real line, but smaller than $Big - M$. In doing so, we interdict the production of A_{min}'s supply on the dummy line. Additionally, we can introduce a preference order between all products regarding the production of the additional demand by differentiating between the respective transportation unit costs to the dummy line. The preference can be e.g. induced by the accumulated over- and underproduction Δp_{mt}.

3.4 Heuristic Staff Planning

The heuristic solution concerning the staff planning on the top level is essentially an anticipation of the decision to be made on the bottom level. This anticipation includes a forecast of the future demand as well as the restrictions concerning the working hours summary. At first we introduce

a preprocessing step determining the production capacity and the minimum number of required employees for every combination of shift model and cycle time for all lines. Incorporating the maximum number of employees to hire in one period and the current staff size, we can determine the number of periods needed to hire enough staff to meet the staff demand in a future period. We observe whether we have to hire additional employees already in the current period which can be seen schematically in Figure 3. The number of periods we look ahead is a parameter with a vital influence on the quality of our anticipation. Next, we have to test whether the working hours

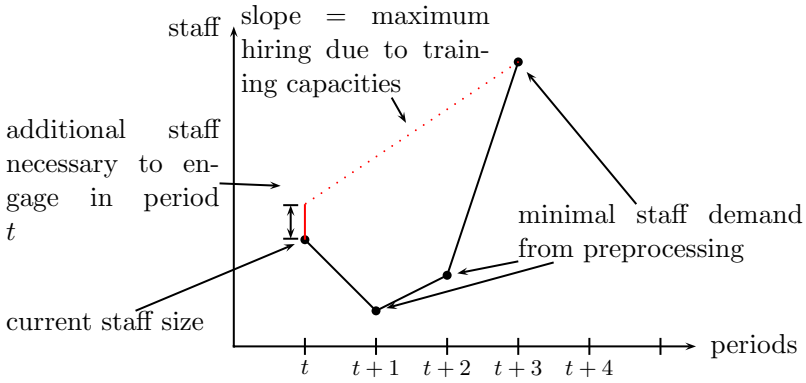


Figure 3: Anticipation of hiring staff for future demand

summary is feasible or not and whether actions are to be taken to improve a critical summary. The summary of a line is the average over all workers employed there. New employees added to the line have a summary of 0. So the average summary is moved towards 0 by new employees, independent of a positive or negative actual summary of the line.

The infeasibility of a working hours summary can be invoked by too many hours of over- or undertime. In the first case additional employees can be hired to bring back the summary into the feasible region. In the second case it is also possible to achieve feasibility by hiring employees in the short term. However, these additional workers cause an oversupply of labor in the following periods so that the summary will become infeasible again which will consequently lead to a vicious circle. For this reason, a state with an infeasible working hours summary caused by undertime is treated as infeasible.

The summary has to reach a total of 0 regularly, so in case of a feasible summary greater than 0 we have to hire additional employees in the following periods. The earlier an employee is hired, the more he can help to reduce the average summary. Hence, we observe that we should generally hire more employees in the current period. Thus, we avoid hiring noticeable more employees in the last periods before the summary has to be 0. If a feasible summary has a negative value, we encounter an analogous problem as described above. Again in a short-term view we can move the summary towards 0 by hiring additional employees but for the long-run the only possibility to reach the value of 0 is to choose a shift model that implies a greater attendance time than the contracted hours. This is outside the scope of this anticipation, so we punish a negative summary with artificial costs in our objective function, e.g. with the value of the accumulated undertime. At this point we know how many permanent and temporary employees we have to dismiss or hire on every line for the production in the current period as well as the surcharge for the future production program and a feasible working hours summary.

Finally, we have to examine whether displacements of permanent staff can avoid hirings. Therefore, we adopt the savings method from the vehicle routing problem and build up three savings lists. In the first list we calculate the savings for displacement instead of dismissing permanent

workers on one line and hiring on another. In the second list the second line has a demand for flexible workers and in the third list the second line does not have a demand for employees but we could dismiss flexible workers on this line and receive permanent ones by displacements to this line. The second and third list are sorted in analogy to the first one. Beginning with the first list and according to the sorting sequence, the displacements are determined where negative savings are also taken into account because it is generally a reasonable assumption to prefer a small staff size.

4 Conclusion

In this paper we presented a Dynamic Programming approach to solve the problem of integrated mid-term production and staff planning in a shop of an automotive plant. We focused on achieving tractability even for the case of parallel production lines that are not independent of each other and disaggregated the solution approach into two levels. On the top level we modeled the problem of distributing of the production workload between the production lines as a transportation problem where the planning of the staff is solved approximately. On the bottom level we fixed all decisions beside the staff planning and determined the optimal staff planning. Areas of future research and development are: First, to integrate the techniques described in this paper into our partner's software tool which is already able to manage subsequent shops each with one production line and intermediate buffers in between them. Second, to improve i.e. to speed up the procedure of the bottom level, e.g. by using a local search approach instead of the Dynamic Programming.

References

[1] R.E. Bellman (1957), *Dynamic Programming*, Princeton University Press, New Jersey.

[2] D.P. Bertsekas (2000), *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont.

[3] L. Delsen, D. Bosworth, H. Groß, and R.M. de Bustillo y Llorente (eds.) (2007), Operating Hours and Working Times, Springer, Berlin.

[4] N. Leopold (1997), *Ein Planungsverfahren zur Kapazitätsbestimmung für Modell-Mix-Montagelinien am Beispiel einer Automobil-Endmontage*, Springer, Berlin.

[5] C. Schneeweiß (1994), Elemente einer Theorie hierarchischer Planung, *OR Spektrum* **16**, 161 – 168.

[6] T.L. Smunt (1986), Incorporate Learning Curve Analysis into medium-term Capacity Planning Prodedures: A Simulation Experiment, *Management Science* **32**(9), 1164 – 1176.

Climbing Depth-bounded Discrepancy Search for Solving Flexible Job Shop Scheduling Problems

Abir Ben Hmida
LAAS-CNRS, Université de Toulouse, Toulouse, France, and ROI, Ecole Polytechnique de Tunisie, La Marsa, Tunisia, abenhmid@laas.fr
Marie-José Huguet, Pierre Lopez
LAAS-CNRS, Université de Toulouse, Toulouse, France, {huguet, lopez}@laas.fr
Mohamed Haouari
ROI, Ecole Polytechnique de Tunisie, La Marsa, Tunisia, mohamed.haouari@ept.rnu.tn

The Flexible Job Shop Scheduling Problem (FJSP) is a generalization of the classical Job Shop Problem in which each operation must be processed on a given machine chosen among a finite subset of candidate machines. The aim is to find an allocation for each operation and to define the sequence of operations on each machine so that the resulting schedule has a minimal completion time. We propose a variant of the climbing discrepancy search approach for solving this problem. Experiments have been performed on well-known benchmarks for flexible job shop scheduling.

Keywords: Scheduling, Allocation, Discrepancy Search, Flexible Job Shop.

1. Introduction

The Flexible Job Shop Problem (FJSP) [1,2] is a generalization of the traditional Job Shop scheduling Problem (JSP), in which it is desired to process a set $J=\{J_1,\ldots,J_n\}$ of n jobs on a set $M=\{1,\ldots,m\}$ of m machines in the shortest amount of time. Every job J_i ($i=1,\ldots,n$) consists of s_i operations $O_{i1}, O_{i2} \ldots, O_{is_i}$ which must be processed in the given order. Every operation O_{is} must be assigned to a unique machine R, selected among a given subset $M_{is} \subseteq M$ ($\forall$ i, $\bigcap_{s=1}^{s_i} M_{is}$ might be nonempty), which must process the operation without interruption during p_i^R units, and a machine can process at most one operation at a time. The goal is to choose for each operation a suitable machine and a starting time so that the maximum completion time C_{max} (the makespan) is minimized
As a generalization of the job shop problem, the FJSP is more complex because of the additional need to determine the assignment of operations to machines. So, this problem is known to be strongly NP-Hard even if each job has at most three operations and there are two machines [3].
Most of the literature on the shop scheduling problem concentrates on the classical JSP case. Relatively recently, the FJSP has captured the interests of many researchers. The first paper that addresses the FJSP was given by Brucker and Schlie [4], which proposes a polynomial algorithm for solving the FJSP with two jobs, in which the processing times are identical whatever the machine chosen to perform an operation. For solving the general case with more than two jobs, two approaches have been used: hierarchical approach and integrated approach. The hierarchical approach is based on the idea of decomposing the original problem in order to reduce its complexity. Brandimarte [5] was the first author to use this decomposition for the FJSP. He solved the assignment problem using some existing dispatching rules and then focused on the resulting job shop subproblems, which are solved using a tabu search heuristic. Mati et al. [6] proposed a greedy heuristic for simultaneously dealing with the assignment and the sequencing subproblems of the flexible job shop model. The advantage of Mati's heuristic is its ability to take into account the assumption of identical machines. Kacem et al. [7] used a genetic algorithm (GA) to solve the FJSP and they adapted two approaches to solve jointly the assignment and the JSP. The first one is to approach by localization. It makes it possible to solve the problem of resource allocation and build an

ideal assignment mode. The second one is an evolutionary approach controlled by the assignment model and applying GA to solve the FJSP.
In this paper, we propose a new discrepancy-based method called *Climbing Depth-bounded Discrepancy Search* (CDDS) to solve the FJSP. Note that CDDS has been first developed to solve Hybrid Flow Shop problems [8] and has proved its efficiency in this domain.
The remainder of this paper is organized as follows. Section 2 introduces the principles of different discrepancy-based methods, as well as their associated algorithms. Section 3 describes the proposed CDDS method while Section 4 presents its performance on Brandimarte's benchmarks. Finally, Section 5 gives some concluding remarks and directions for future work.

2. Discrepancy-based search methods

Discrepancy-based methods are tree search methods developed for solving hard combinatorial problems. These methods consider a branching scheme based on the concept of discrepancy to expand the search tree. This can be viewed as an alternative to the branching scheme used in a Chronological Backtracking method. The primal method, Limited Discrepancy Search (LDS), is instantiated to generate several variants, among them, Depth-bounded Discrepancy Search (DDS) and Climbing Discrepancy Search (CDS).

2.1 Limited Discrepancy Search

The objective of LDS proposed by Harvey in [9] is to provide a tree search method for supervising the application of some ordering heuristics (variable and value ordering). It starts from initial variable instantiations suggested by a given heuristic and successively explores branches with increasing discrepancies from it, i.e., by changing the instantiation of some variables. Basically, this number of changes corresponds to the number of discrepancies from the initial instantiation. The method stops when a solution is found (if such a solution does exist) or when an inconsistency is detected (the tree is entirely expanded).
The concept of discrepancy was first introduced for binary variables. In this case, exploring the branch corresponding to the best Boolean value (according a value ordering) involves no discrepancy while exploring the remaining branch implies just one discrepancy. It was then adapted to suit to non-binary variables in two ways (Figure 1). The first one considers that choosing the first ranked value (rank 1) leads to 0 discrepancy while choosing all other ranked values implies 1 discrepancy. In the second way, choosing value with rank r implies r–1 discrepancies. In our work, we chose to adopt the first way for the discrepancy-counting.

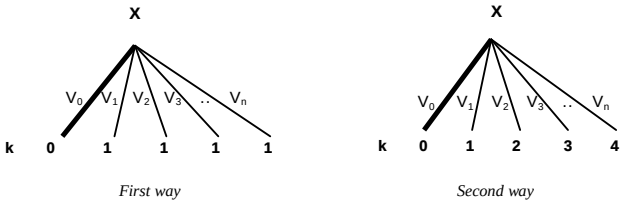


Figure 1. Counting discrepancies

Dealing with a problem defined over N binary variables, an LDS strategy can be described as shown in Algorithm 1. In such a primal implementation, the main drawback of LDS is to be highly redundant: during the search for solutions with k discrepancies, solutions with 0 to k–1 discrepancies are revisited. To avoid this, Improved LDS method (ILDS) was proposed in [10]. Another improvement of LDS consists in applying discrepancy first at the top of the tree to correct early mistakes in the instantiation heuristic; this yields the Depth-bounded Discrepancy Search method (DDS) proposed in [11]. In the DDS algorithm, a given depth limits the generation of leaves with k discrepancies.
All these methods (LDS, ILDS, DDS) lead to a feasible solution, if it exists, and are closely connected to an efficient instantiation heuristic. These methods can be improved by adding local con-

straint propagation such as Forward Checking [12]. After each instantiation, Forward Checking suppresses inconsistent values in the domain of not yet instantiated variables involved in a constraint with the assigned variable.

```
k ← 0    -- k is the number of discrepancies
k_max ← N  -- N is the number of variables
Sol ← Initial_solution()   -- Sol is the reference solution
while No_Solution() and (k < k_max) do
    k ← k+1
    -- Generate leaves at discrepancy k from Sol
    -- Stop when a solution is found
    Sol' ← Compute_Leaves(Sol, k)
    Sol ← Sol'
end while
```

Algorithm 1. Limited Discrepancy Search

2.2 Climbing Discrepancy Search

CDS is a local search method that adapts the concept of discrepancy to find a good solution for combinatorial optimization problems [13]. It starts from an initial solution suggested by a given heuristic. Hence nodes with discrepancy equal to 1 are first explored then those having a discrepancy equal to 2, and so on. When a leaf with an improved value of the objective function is found, the reference solution is updated, the number of discrepancies is reset to 0, and the process for exploring the neighborhood is again restarted (see Algorithm 2).

```
k ← 0    -- k is the number of discrepancies
k_max ← N  -- N is the number of variables
Sol ← Initial_Solution() -- Sol is the reference solution
while (k < k_max) do
    k ← k+1
    -- Generate leaves at discrepancy k from Sol
    Sol' ← Compute_Leaves(Sol, k)
    if Better(Sol', Sol) then
          -- Update the current solution
          Sol ← Sol'
          k ← 0
    end if
end while
```

Algorithm 2. Climbing Discrepancy Search

The aim of CDS strategy is not only to find a feasible solution, but rather a high-quality solution in terms of criterion value. As mentioned by their authors, the CDS method is close to the Variable Neighborhood Search (VNS) [14]. VNS starts with an initial solution and iteratively explores neighborhoods more and more distant from this solution. The exploration of each neighborhood terminates by returning the best solution it contains. If this solution improves the current one, it becomes the reference solution and the process is restarted. The interest of CDS is that the principle of discrepancy defines neighborhoods as branches in a search tree. Thus, a gradual increase of the allowed discrepancies builds variable-size neighborhoods. The use of a discrepancy-based procedure therefore leads to structure the local search method and then to restrict redundancies.

3. The proposed approach

3.1 Problem variables and constraints

To solve the FJSP under study, we have to select an operation, to allocate a resource for the selected operation, and to set its start time. To reduce the makespan, we only consider two kinds of variables: operation selection and resource allocation. (The start time of each operation will be set at the earliest possible value). The values of these two types of variables are ordered following a given instantiation heuristic presented below.

We denote by X the operation selection variables vector and by A the resource allocation variables vector. Thus, X_i corresponds to the i^{th} operation in the sequence and A_i is its affectation value ($\forall i = 1,..., N$, with N the number of all operations). The domain of variable X_i is $\{O_{11}, O_{12},\ldots, O_{1s_1}, O_{21},\ldots, O_{n1},\ldots, O_{ns_n}\}$ which corresponds to the choice of the operation to be scheduled. The values taken by the X_i variables have to be all different. The A_i domains are $\{1,\ldots,m\}$. Moreover, we consider precedence constraints between two consecutive operations of the same job and precedence constraints that can join each operation with other jobs operations.

3.2 Discrepancy for flexible job shop problems

Because of the existence of two types of variables, we consider here two types of discrepancies: discrepancy on operation selection variables and discrepancy on resource allocation variables. Indeed, our goal is to improve the makespan of our solutions, and since resources are not identical, discrepancy on allocation variables can improve it. Also, the sequence of jobs to be scheduled may have an impact on the total completion time.
Therefore, achieving a discrepancy consists in:

- Selecting another operation to be scheduled than the operation given by a value ordering heuristic. Operation selection variables are N-ary variables. The number of discrepancy is computed as follows: the first value given by the heuristic corresponds to 0 discrepancy, all the other values correspond to 1 discrepancy. Let consider 3 operations and let consider that the first selected operation is O_1, O_2 the second, and O_3 the third one (this order is given by a value ordering heuristic). Selecting another operation than O_1 in the first position (X_1) consists in making a discrepancy in this level, and so on (see Figure 2).

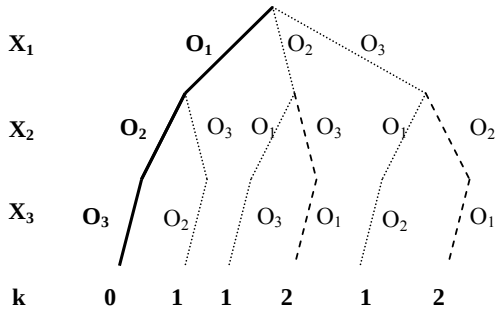


Figure 2. Discrepancies only on the three first operation selection variables.

- Assigning the operation to another resource than the resource suggested by a value ordering heuristic. The number of discrepancy is computed as follows: the first value given by the heuristic corresponds to 0 discrepancy, all the other values correspond to 1 discrepancy. In this case, let consider that O_1 can be processed by one machine among the set {R_2, R_1, R_3 (this order is given by a value ordering heuristic)}, O_2 by {R_1, R_4}, and O_3 by only R_1. Selecting another machine than R_2 for the first operation (O_1) consists in making a discrepancy in this level, and so on (see Figure 3).

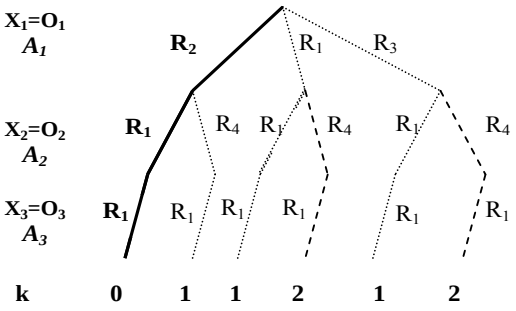


Figure 3. Discrepancies only on the three first resource allocation variables.

To obtain solutions of k+1 discrepancies directly from a set of solutions with k discrepancies (without revisiting solutions with 0,…, k-1 discrepancies), we consider for each solution the last instantiated variable having the k^{th} discrepancy value and we just have to choose a remaining variable for the $k+1^{th}$ discrepancy value.

3.3 Instantiation heuristics and propagation

The exploration strategy first consider operation selection variable to choose an operation, secondly consider resource allocation variable to assign the selected operation to a resource.
We have two types of value ordering heuristics: the first one ranks operations whilst the second one ranks resources.

Type 1: Operation selection. Several priority lists have been used. We first give the priority to the operation with the earliest start time (EST) and, in case of equality, we consider three alternative rules:

- SPT (Smallest Processing Time) rule. It orders the operations in the queue in the ascending order of processing times. When a machine is free, the next operation with the shortest time in the queue will be removed for processing.
- EDD (Earliest Due Date) rule. It gives the priority to the operation which belongs to the job with the earliest due date.
- CJ (Critical Job) rule. It gives the priority to the operation belonging to the job with the longest total duration.

Type 2: Assignment of operations to machines. The operation of the job chosen by the heuristic of Type 1 is assigned to the machine such that the operation completes as soon as possible; hence, following an Earliest Completion Time (ECT) rule. This latter rule is dynamic, that is, the machine with the highest priority depends on the machines previously loaded.

After each instantiation of Type 2, we use a Forward Checking constraint propagation mechanism to update the finishing time of the selected operation as well as the starting time of the successor operation. We also maintain the availability date of the chosen resource.

3.4 Proposed discrepancy-based method

In our problem, the initial leaf (with 0 discrepancy) is a solution since we do not constrain the makespan value. Nevertheless, we may use the discrepancy principle to expand the tree search for visiting the neighborhood of this initial solution. The only way to stop this exploration is to set a limit for the CPU time or to reach a given lower bound on the makespan. To limit the search tree, one can use the DDS method that considers in priority variables at the top of the tree (job selection at the initial stages).
To improve the search, we have to consider the CDS method that goes from an initial solution to a better one, and so on. The idea of applying discrepancies only at the top of the search tree can be also joined with CDS algorithm to limit the tree search expansion. So, we have developed a new strategy called *Climbing Depth-bounded Discrepancy Search* (CDDS) [8]. With this new method,

one can restrict neighborhoods to be visited by only using discrepancies on variables at the top of the tree (see Algorithm 3).

```
k ←0     -- k is the number of discrepancy
k_max ← N  -- N is the number of variables
Sol ← Initial_Solution() -- Sol is the reference solution
while (k < k_max) do
    k ← k +1
    -- Generate leaves at discrepancy k from Sol
    -- and at d-depth value from the top of the tree with 1 < d < N
    Sol'← Compute_Leaves(Sol, d, k)
    if Better(Sol', Sol) then
          -- Update the current solution
          Sol ← Sol'
          k ← 0
    end if
end while
```

Algorithm 3. Climbing Depth-bounded Discrepancy Search

The next section is devoted to the evaluation of this algorithm for flexible job shop scheduling.

4. Computational experiments

4.1 Test beds

We have compared our proposed CDDS method for solving a set of 10 benchmarks instances presenting by Brandimarte in [5]. In [1], all the problems have been solved using a Tabu Search (TS) method.

We propose to solve problems under study by the use of three different strategies for applying discrepancies:

S1- Considering discrepancy only on operation selection variables.

S2- Considering discrepancy only on resource allocation variables.

S3- Join the two kinds of discrepancies. Here we consider the best solution given by S1 as a reference solution to S2.

In our study, we propose to compare these three strategies in terms of solutions quality. Also, we report a comparison between the three heuristics (SPT, CJ and EDD).

Next, we have tested our proposed CDDS method for solving another class of instances presented in [16]. In this class of instances resources are identical. Hence, doing a discrepancy on resource allocation variables does not have any interest. We therefore propose to solve those problems using strategy S1 only. This class of instances contains three sub-classes: the first one, noted Edata, which few number of operations that can be assigned to different machines. The second one, noted Rdata, which most operations can be assigned to a few number of different machines, and the last one, noted Vdata, which each operation can be assigned to many different machines.

We have set a maximum CPU time limit to 1800 s. If no optimal solution was found within 1800 s, then the search is stopped and the best solution of CDDS is output as the final schedule. The depth of discrepancy in our methods varies between 3 and 8 from the top of the tree. We have carried out our tests on a Pentium IV 3.20 GHz with 448 Mo RAM. The CDDS algorithm has been coded using C language and run under Windows XP Professional.

4.2. Computational results

In Table 1, for all considered problems, we present the best makespan values obtained by CDDS method among the different value ordering heuristics (SPT, CJ and EDD) and among the different strategies, and the TS algorithm of [1]. We also, present the CPU time of the best obtained makespan values. CDDS returns the best known solutions, denoted by an asterisk (*), for 60% of the ten instances. If all instances are considered, the average deviation of CDDS algorithm from the best known solutions is 1.4%. Table 1 gives, also, a comparison between the three strategies

(S1, S2, and S3) for discrepancies. We consider the best makespan values obtained by each strategy among the different value ordering heuristics (SPT, CJ, and EDD). The third strategy (S3) always gives better solutions in a fixed running time. This result is not surprising since the latter strategy combines the two types of discrepancies.

problems	*Job×Mach*	*LB*	*UB*	*S1*	S2	*S3*	*CDDS*	Av_CDDS (CPU)	TS	Av_TS (CPU)
mk01	10×6	36	42	42	43	40*	40*	0.5	40	0.01
mk02	10×6	24	32	28	30	26*	26*	12.7	26	0.73
mk03	15×8	204	211	204*	204*	204*	204*	0	204	0.01
mk04	15×8	48	81	69	68	60*	60*	136	60	0.08
mk05	15×4	168	186	181	183	175	175	9.6	173	0.96
mk06	10×15	33	86	67	70	60	60	152	58	3.26
mk07	20×5	133	157	148	160	144*	144*	132	144	8.91
mk08	20×10	523	523	526	531	523*	523*	20	523	0.02
mk09	20×10	299	369	331	344	308	308	89	307	0.15
mk10	20×15	165	296	229	240	216	216	7.03	198	7.69

Table 1.Results on Brandimarte's data.

Table 2 shows computational results over two instance classes. The first column reports the data set, the second column the number of instances for each class, the third column the average number of alternative machines per operation. The next three columns respectively report the deviation percentage of the best solution obtained by TS [1], by CDDS, and by the best known genetic algorithm (GA) solving these problems [19], with respect to the best known lower bound. The table shows that our algorithm is stronger with a higher degree of flexibility (Hurink VData). Furthermore, the results show that our algorithm outperforms the best genetic algorithm (still remaining less efficient than TS.

Data set	*num*	*alt*	*TS (%)*	*CDDS (%)*	*GA (%)*
Brandimarte	10	2.59	15.4	17.3	17.5
Hurink Edata	43	1.15	2.2	5.3	6.0
Hurink Rdata	43	2	1.2	2.5	4.4
Hurink Vdata	43	4.31	0.1	0.6	2.0

Table 2.Deviation percentage over the best known lower bound

Table 3 presents a comparison between rules (SPT, CJ, and EDD) performances. We consider the best makespan values obtained by each ordering heuristic. The third rule (EDD) always gives better solutions in a fixed running time.

	%Deviation from LBs		
heuristics	*SPT*	*CJ*	*EDD*
Brandimarte	28.2	26.2	17.9
Hurink Edata	16.7	13.9	13.6
Hurink Rdata	13.7	6.9	5.7
Hurink Vdata	3.3	1.2	1.1

Table 3. Performances of search heuristics

5. Conclusion

In this paper a Climbing Depth-bounded Discrepancy Search (CDDS) method is presented to solve a Flexible Job Shop Scheduling Problem with the objective of minimizing makespan. Our CDDS approach is based on ordering heuristics and involves two types of discrepancies: operation selection and resource allocation. The test problems are benchmarks used in the literature. Our results are not better compared with those obtained using a Tabu Search, but in terms of makespan, we

can consider that the CDDS method provides promising results, especially if we consider Hurink's instances. Results show that our algorithm is more efficient with a higher degree of flexibility (Hurink VData). Furthermore, the results show that our algorithm outperforms the best genetic algorithm but it remains less efficient than TS for the two instance classes. The percentage deviations from these latter results are presented.
Developments can still be done to improve the solution's quality of CDDS algorithm. Moreover, other variants of CDDS algorithm may be envisaged for instance by including efficient lower bounds for the FJSP, as well as heuristics for backjumping on promising choice points (see [15]). Other types of problems can be also considered like problems presented in [17,18], which present different properties in terms of total and partial flexibility. Experiments on these latter problems are still in progress.

References

[1] M. Mastrolilli, L. M. Gambardella (2000), Effective neighbourhood functions for the flexible job shop problem, *Journal of Scheduling* **3**, 3 – 20.
[2] R. Vaessens (1995), *Generalized Job Shop Scheduling: Complexity and Local Search,* PhD thesis, Eindhoven University of Technology.
[3] E. Lawler, J.K. Lenstra, A. Rinnooy Kan, D. Shmoys (1975). Sequencing and scheduling: Algorithms and complexity. *Handbook in Operations Research and Management Sc.* **4**, 115 – 124.
[4] P. Brucker, R. Schlie (1990), Job shop scheduling with multi-purpose machines, *Computing* **45**, 369 – 375.
[5] P. Brandimarte (1993), Routing and scheduling in a flexible job shop by tabu search, *AOR* **41**, 157 – 183.
[6] Y. Mati, N. Rezg, X. Xie (2001), An integrated greedy heuristic for a flexible job shop scheduling problem, *Proceedings IEEE-SMC'01*, 2534 – 2539.
[7] I. Kacem, S. Hammadi, P. Borne (2002), Approach by localization and multiobjective evolutionary optimization for flexible job shop scheduling problems, *IEEE Transactions on Systems, Man and Cybernetics, Part C* **32**(1), 408 – 419.
[8] A. Ben Hmida, M.-J. Huguet, P. Lopez, M. Haouari (2006), Adaptation of Discrepancy-based Methods for Solving Hybrid Flow Shop Problems, *Proceedings IEEE-ICSSSM'06*, 1120 – 1125.
[9] W.D. Harvey (1995), *Nonsystematic backtracking search*, PhD thesis, CIRL, Univ. of Oregon.
[10] R.E. Korf (1996), Improved Limited Discrepancy Search, *Proceedings AAAI-96*, 286 – 291.
[11] T. Walsh (1997), Depth-bounded Discrepancy Search, *Proceedings IJCAI-97*, 1388 – 1395.
[12] R. Haralick, G. Elliot (1980), Increasing tree search efficiency for constraint satisfaction problems, *AI* **14**, 263 – 313.
[13] M. Milano, A. Roli (2002), On the relation between complete and incomplete search: an in formal discussion, *Proceedings CPAIOR'02*, 237 – 250.
[14] P. Hansen, N. Mladenovic (2001), Variable neighborhood search: Principles and applications, *EJOR* **130**, 449 – 467.
[15] M.-J. Huguet, P. Lopez, A. Ben Hmida (2004), A limited discrepancy search method for solving disjunctive scheduling problems with resource flexibility, *Proceedings PMS'04*, 299–302.
[16] E. Hurink, B. Jurisch, M. Thole (1994), Tabu search for the job shop scheduling problem with multi-purpose machines, *Operations Research Spectrum* **15**, 205 – 215.
[17] S. Dauzère-Pérès, J. Paulli (1997), An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search, *AOR* **70**, 281 – 306.
[18] J.W. Barnes, J.B. Chambers (1996), Flexible job shop scheduling by tabu search, Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-09, http://www.cs.utexas.edu/users/jbc/.
[19] F. Pezzella, G. Morganti, G. Ciaschetti (2007), A genetic algorithm for the flexible job-shop scheduling problem, *Computers & Operations Research*, Article in Press, Corrected Proof, http://dx.doi.org/10.1016/j.cor.2007.02.014.

Complex Job Shop Multiple Orders per Job Scheduling

Jagadish Jampani, Scott J. Mason
University of Arkansas, 4207 Bell Engineering Center, Fayetteville, AR 72701 USA,
{jjampan, mason}@uark.edu

Semiconductor manufacturing production scheduling is a challenging task due to process complexity that includes the assignment of customer orders to front opening unified pods (FOUPs), FOUP batch processing, parallel machines, and re-entrant flow. A network-based optimization model and a column generation (CG) heuristic are presented for this problem to minimize the weighted completion time of customer orders. Computational results show that the CG heuristic obtains solutions that are very close to/better than a mixed-integer program-based heuristic (MIP heuristic) for problem instances without order ready times. In the problem instances with order ready times, CG solutions are within 11% of MIP heuristic solutions, but are obtained in seconds rather than hours.

Keywords: semiconductor manufacturing, complex job shop, column generation

1. Introduction

Semiconductor manufacturing is a complex, expensive manufacturing process involving multiple process flows typically containing hundreds of process steps each. Wafer fabrication is the most complex stage in the integrated circuit (IC) manufacturing process [11]. Customers place orders for ICs, hundreds of which can be fabricated on a single silicon wafer. The newest wafer fabs manufacture ICs on silicon wafers 300-mm in diameter. Given the size/weight and value of a 300-mm wafer containing ICs, front opening unified pods (FOUPs) are used to store and to transport wafers between workstations in 300-mm wafer fabs in groups of 13 or 25. FOUPs are filled with inert gas to protect against contaminants/particulates coming into contact with the wafer surface. Customer IC orders are converted into equivalent 300-mm silicon wafer starts. As an order may require only 3-4 wafers to fill, multiple customer orders are combined into a single FOUP for fab processing and automated material handling system transport. Although each customer order has unique attributes (size, ready time, weight, promise date, etc.), 300-mm fab scheduling is performed at the FOUP (job) level. This is known as multiple orders per job (*moj*) scheduling [10].

A semiconductor wafer fab is a complex job shop (CJS) [5], as it contains parallel machines operating in *tool groups*, batch processes, and re-entrant process flows. In addition, tool groups may contain sequence-dependent setup times (e.g., ion implanters). Scheduling 300-mm wafer fabs adds an additional layer of complexity, as FOUPs contain multiple customer orders that may or may not remain together in the same FOUP throughout the entire manufacturing process flow. In this paper, we investigate the *moj* CJS scheduling problem ("*moj*-CJSSP") to minimize total weighted order completion time for the case when a FOUP's processing time is independent of its contents (i.e., *single lot* in [10]). In $\alpha|\beta|\gamma$ scheduling notation [3], the *moj*-CJSSP of interest can be denoted as $FJc|moj(lot),r_o,recrc|\sum w_oC_o$, where w_o (C_o) denotes order *o*'s weight (completion time). The *moj*-CJSSP is NP hard, as the NP hard $Jm||C_{\max}$ reduces to it.

Ovacik and Uzsoy [9] develop a CJS scheduling heuristic by using shop floor status information, but only consider re-entrant flows. Mason *et al.* [5] present a modified Shifting Bottleneck heuristic (MSBH) for minimizing total weighted tardiness (TWT) in a complex job shop, although a subsequent cycle-elimination procedure is required to guarantee feasibility [6]. Mason *et al.* [7] then present a mixed-integer program (MIP) for TWT minimization in a CJS and compare it to the MSBH and three dispatching rules. Moench and Driessel [8] present a distributed SBH to address minimizing TWT for the CJSSP. The overall problem is decomposed into two hierarchical stages: an upper stage decides start dates and planned due dates, while the lower stage uses upper stage inputs in a SBH. While Jampani and Mason [4] present column generation (CG) approaches for parallel machine *moj* scheduling problems, the *moj*-CJSSP has not been addressed in the literature.

2. Solution Approaches for *moj*-CJSSP

2.1. Network-Based MIP Formulation

Modelling MIPs using a network flow-based paradigm can be more computationally efficient than classical disjunctive approaches. The following steps create the network structure for the MIP formulation. Initially the steps followed to create a network representation for the *moj*-CJSSP corresponding to a single tool group is presented. This network structure is then extended to incorporate multiple tool groups and re-entrant flows.

Step 1: A dummy source node has a branch for each machine in the tool group. The upper bound of flow on corresponding arcs is equal to the number of orders and the lower bound is zero. This gives the flexibility of assigning any order to any machine in the tool group.

Step 2: Each of the nodes representing a machine branches out with arcs equal to the number of batches that can be processed on that machine. The upper bound of the flow on the corresponding arcs is equal to the number of orders and the lower bound is zero. Hence, any number of orders can be assigned to a batch subject to the capacity constraints.

Step 3: Each of the nodes representing a batch branches out into nodes equal to the batch size or the number of FOUPs that can be assigned to that batch in that tool group. The number of arcs connecting a head node to the tail node at this FOUP level is equal to the number of orders. The upper bound of flow on the corresponding arcs is equal to one and the lower bound is zero. By having the binary variables at this level, an order is either selected or not and orders will not be partially assigned to multiple FOUPs.

Step 4: All the nodes at the FOUP level are connected to one dummy sink node. Each FOUP node is connected to the sink node using only one arc. The upper bound on these arcs is equal to the number of orders and the lower bound is zero.

The path selected by an order gives all requisite assignment information for that order. Hence, all the arcs represent the decision variables in the formulation, which are non-continuous variables. With the information obtained regarding the path followed by an order in the network corresponding to a tool group, it can be traced back to find the order to FOUP, FOUP to batch, batch to machine, order to batch and order to machine assignments in that tool group. In this network, FOUP-to-batch and batch-to-machine assignments are done inherently. Hence, the overall assignments boil down to assigning orders to FOUPs subject to the network continuity constraints. This helps in cutting down significant number of constraints and binary variables in the formulation.

An example is presented to explain the steps involved in this network creation. A *moj* problem with ten orders and a tool group with two identical machines are considered. Each machine can accommodate two batches and two FOUPs can be assigned to each of the batches.

Step 1: As two machines are present in the tool group, the dummy source node has arcs 1 and 2 as shown in Figure 1. The upper bound on arcs 1 and 2 is 10 for the considered 10 orders problem instance and the corresponding lower bound is zero.

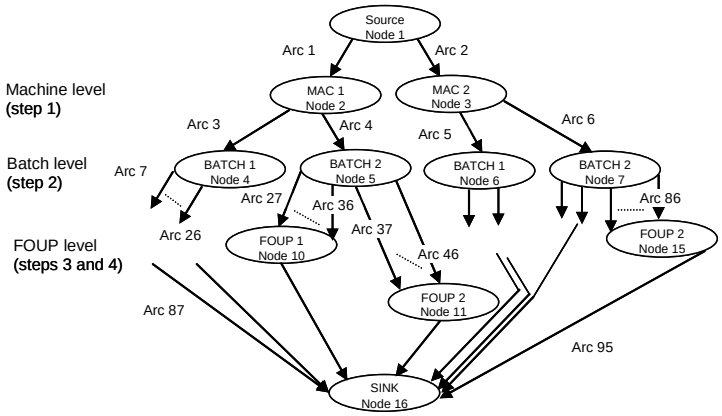


Figure 1. Example problem for network construction

Step 2: As two batches can be processed on each of the machines, the node representing machine 1 branches into 3 and 4, and similarly node 2 branches into arcs 5 and 6. The upper bound on arcs 3 through 6 is 10 in this example and the corresponding lower bound is zero.

Step 3: As two FOUPs can be assigned to a batch, FOUPs in batch 1 on machine 1 are represented by nodes 8 and 9. Similarly, FOUPs in batch 2 on machine 1 are represented by nodes 10 and 11. The number of arcs connecting node 5 and node 10 is equal to the number of orders (i.e., 10). The upper bound on each arcs at the FOUP level=1 and the lower bound=0.

Step 4: In this step, a new node 16 is created and the nodes from the FOUP level (nodes 8 through 15) are connected to sink node 16.

Network creation steps 1-4 are repeated to create a network with multiple tool groups. The dummy source node for the succeeding tool group is the sink node of the previous tool group. In case of reentrant flow, an order needs to be processed in the same tool group multiple times. This is accomplished by creating a duplicate tool group and restricting the machines of the tool group to process only one batch at a time. For example, if tool group 2 is a duplicate of tool group 1 containing a single machine, then the machine can process a batch either in tool group 1 or 2 at a time satisfying the machine capacity constraint. The following notation is used in our MIP formulation:

o, j, b, m, p	Index for orders, FOUP #, batch #, machine #, and process step #, respectively
K	FOUP capacity
s_o, r_o, w_o	Size, ready time and weight of order *o*, respectively
ρ_p, c_p	Processing time and tool group # for process step *p*, respectively
$J(c_p)$	Set of FOUPs in tool group c_p
$B(c_p)$	Set of batches in tool group c_p
$M(c_p)$	Set of machines in tool group c_p
$X_{o,j,b,m,p}$	Variable = 1 if order *o* is assigned to FOUP *j* that is present in batch *b* and processed on machine *m* at process step *p*; otherwise, = 0.
$Y_{b,m,p}$	Variable for completion time of batch *b* processed on machine *m* at process step *p*
$Z_{b,m,p}$	Variable for ready time of batch *b* processed on machine *m* at process step *p*
$R_{o,p}$	Variable for ready time of order *o* at process step *p*
$F_{o,p}$	Variable for completion time of order *o* at process step *p*

The formulation for $FJc|moj(lot), r_o, recrc|\sum w_o C_o$ problem is as follows:

Minimize $$\sum_{o\in O} w_o F_{o,p=P} \tag{1}$$

Subject to:
$$\sum_{m\in M(c_p)} \sum_{b\in B(c_p)} \sum_{j\in J(c_p)} X_{o,j,b,m,p} = 1 \qquad \forall o \in O, p \in P \tag{2}$$

$$\sum_{o\in O} s_o X_{o,j,b,m,p} \le K \qquad \forall j \in J(c_p), b \in B(c_p), m \in M(c_p), p \in P \tag{3}$$

$$Z_{b,m,p} \ge r_o X_{o,j,b,m,p} \qquad \begin{array}{l}\forall o \in O, j \in J(c_p), b \in B(c_p) \\ \forall m \in M(c_p), p = 1\end{array} \tag{4}$$

$$Y_{b,m,p} - Z_{b,m,p} \ge \rho_p \qquad \forall b \in B(c_p), m \in M(c_p), p \in P \tag{5}$$

$$Z_{b,m,p} \ge F_{o,p-1} - M_{big}(1 - X_{o,j,b,m,p}) \qquad \begin{array}{l}\forall o \in O, j \in J(c_p), b \in B(c_p), \\ m \in M(c_p), p > 1, p < P\end{array} \tag{6}$$

$$F_{o,p} \ge Y_{b,m,p} - M_{big}(1 - X_{o,j,b,m,p}) \qquad \begin{array}{l}\forall o \in O, j \in J(c_p), b \in B(c_p), \\ m \in M(c_p), p \in P\end{array} \tag{7}$$

$$Z_{b,m,p} \geq Y_{b-1,m,p} \qquad \forall b \in B(c_p), b > 1, m \in M(c_p), p \in P \qquad (8)$$

$$X_{o,j,b,m,p} \in \{0,1\} \qquad \forall o \in O, j \in J(c_p), b \in B(c_p), m \in M(c_p), p \in P \qquad (9)$$

Objective function (1) minimizes the sum of weighted completion times of the orders during the last process step. Constraint set (2) makes sure that every order is assigned to a FOUP during every process step. Constraint set (3) corresponds to FOUP capacity constraints. Orders are assumed to have non-zero ready times. Constraint set (4) calculates batch ready times as a function of order ready times. Constraint set (5) takes care of the processing time of the orders in each of the tool groups during the process steps. During each process step, the ready time of the batches is dependent on the orders assigned to the batches and the orders completion time during the immediate preceding process step. Ready time of the batches is calculated at each process step by constraint set (6). Constraint set (7) calculates the completion time of the orders at each process step. Constraint set (8) restricts the starting time of batch *b* to be at least the completion time of batch $b-1$. Hence, it also restricts only one batch to be processed on a machine at a time.

Constraint sets (2) to (8) correspond to assignment and ready time constraints. As our *moj*-CJSSP formulation is network-based, the network is stored in terms of arc details, which are head/tail nodes and lower/upper bounds on the arcs. These arcs represent decision variables in the formulation. Network constraints pertain to upper and lower arc capacity restrictions, as well as arcs flow and flow continuity at nodes. The following notation is used in this part of the formulation:

g, n	Index over the arcs $(g \in G)$ and nodes $(n \in N)$, respectively
d, f	Index over the tail and head nodes, respectively
$\omega(n)$	Set of tail nodes of the arcs that have *n* as their head node
$\xi(n)$	Set of head nodes of the arcs that have *n* as their tail node
θ, ψ	Source and sink nodes of the network, respectively
$U_{d,f}$	Variable for amount of flow on an arc with tail node *d* and head node *f*

Additional network constraints for the $FJc|moj(lot), r_o, recrc|\sum w_o C_o$ formulation are as follows:

$$\sum_{f \in \xi(\theta)} U_{d=\theta,f} = O \qquad (10)$$

$$\sum_{d \in \omega(\psi)} U_{d,f=\psi} = O \qquad (11)$$

$$\sum_{d \in \omega(n)} U_{d,n} - \sum_{f \in \xi(n)} U_{n,f} = 0 \qquad \forall n \in N - \{\psi, \theta\} \qquad (12)$$

Constraint sets (10) and (11) require the sum of arc flows out of the source and into the sink equal the number of orders, respectively. Constraint set (12) maintains flow balance at each non-source and non-sink node. Depending on what an arc represents, its upper bound is set and the lower bound of all the arcs is zero. If an arc corresponds to the FOUP level (Figure 1), the upper bound on the corresponding binary decision variable is one unit. Otherwise, the upper bound on the decision variable is the number of orders. As the arcs at the FOUP level represent binary decision variables, considering flow continuity constraints results in the other arcs taking on only integer values. Therefore, they are declared as continuous variables to reduce MIP computation time. The following valid inequalities also help in improving the lower bound for this minimization problem and thereby reducing computation time.

$$F_{o,p} \geq \rho_p \qquad o \in O, p = 1 \qquad (13)$$

$$F_{o,p} \geq F_{o,p-1} + \rho_p \qquad o \in O, p > 1 \qquad (14)$$

$$G_{o,m,p} \geq X_{o,j,b,m,p} \qquad \forall o \in O, j \in J, b \in B(c_p), \forall m \in M(c_p), p \in P \qquad (15)$$

$$T_{m,p} \geq F_{o,p} - M(1 - G_{o,m,p}) \qquad \forall o \in O, m \in M(c_p), p \in P \qquad (16)$$

$$Z_{b,m,p} \geq T_{m,p} \qquad \forall b \in B(c_p), m \in M(c_p), p \in P \tag{17}$$

Constraint set (13) restricts the completion time of an order during the first process step to be at least equal to its processing time. Completion time of an order in any process step ($p > 1$) is at least equal to the sum of processing time in the current process step and completion time of the order in the immediate previous process step. This condition is taken care of by constraint set (14). Constraint set (15) finds the machine on which an order is processed during a particular process step. Constraint set (16) calculates maximum completion time of processing all the orders on a particular machine during a process step. Constraint set (17) requires the ready time of an order at a tool group to be at least the completion time of the same order's previous visit to the same tool group, where *p* represents re-entrant flow steps only.

2.2 Column Generation Heuristic

It can be observed that the MIP can be decomposed into two problems, as in the case of CG. The column generation approach is based on the idea of *decentralized decision processes* [1]. In CG, the linear program is decomposed into multiple sub-problems and a master problem—the master has global visibility and coordinates linear transformations of vectors obtained from the sub-problems. The master problem (MP) selects the best feasible columns, while the subproblem generates new columns representing the assignment information corresponding to the process steps in the complex job shop. As the critical part of any CG approach is modeling the subproblem, a network-based subproblem is developed that is similar to the network created in Section 2.1.1. This results in more efficient subproblem computations that directly result in reduced overall computation time. The following notation is used in the CG formulation for the *moj*-CJSSP:

o	Index over the order number ($o \in O$)
k	Index over the slot number in a FOUP (*K* is FOUP capacity) ($k \in K$)
t_n	Index over the FOUP number during process step *n* ($t_n \in T_n$)
m_n	Index over the machine number during process step *n* ($m_n \in M_n$)
c	Index over the columns or patterns
n	Index over the process steps in the complex job shop problem ($n \in N$)
$b(n)$	Batch size during process step *n*
C(o)	Columns corresponding to order *o* ($c \in C(o)$)
$W_{o,c}$	Product of w_o and completion time of order *o* in the last process step, which is represented in column *c*
$A_{o,k,t_n,m_n,c}$	Variable = 1, if order *o* occupying slot *k* in FOUP t_n is processed on machine m_n and this information is represented in column *c*; otherwise, = 0
E	Maximum number of FOUPs that can be used
M_{big}	Represents a large positive number (i.e., big *M*)
$X_{o,c}$	Variable = 1 if column *c* corresponding to order *o* is selected; otherwise, = 0
Z_{t_n,m_n}	Variable = 1 if FOUP *t* is occupied and processed on machine m_n; otherwise, = 0
λ	Variable to find how many extra FOUPs are used than expected

Restricted Master Problem (MP)

$$\text{Minimize } M_{big}\lambda + \sum_{o \in O}\sum_{c \in C(o)} W_{o,c}X_{o,c} \tag{18}$$

$$\text{Subject to: } \sum_{n \in N}\sum_{m_n \in M_n}\sum_{t_n \in T_n} Z_{t_n,m_n} - \lambda \leq E \tag{19}$$

$$\sum_{o \in O}\sum_{c \in C(o)}\sum_{k \in K} A_{o,k,t_n,m_n,c} - M_{big}Z_{t_n,m_n} \leq 0 \;\; \forall t_n \in T_n, m_n \in M_n, n \in N \tag{20}$$

$$\sum_{c \in C(o)} X_{o,c} = 1 \qquad \forall o \in O \tag{21}$$

$$\sum_{o \in O} \sum_{c \in C(o)} A_{o,k,t_n,m_n,c} X_{o,c} \le b(n) \qquad \forall t_n \in T_n, k \in K, m_n \in M_n, n \in N \tag{22}$$

$$X_{o,c} \ge 0 \qquad \forall o \in O, c \in C \tag{23}$$

The MP objective function (18) minimizes total weighted order completion time and penalizes the use of more FOUPs than are available. Constraint sets (19) and (20) assess whether the number of FOUPs used exceeds *E*, the maximum number allowed. Constraint set (21) selects a column corresponding to every order once. Constraint set (22) enforces batch capacity restrictions for each machine in the CJS. This constraint also controls FOUP to batch assignment. This formulation assumes batches are processed sequentially on a machine in increasing order of batch index without forced idle time. The following notation is used in the CG subproblem (SP) of the *moj*-CJSSP:

∂, $\sigma_{t,m_n,n}$ Dual values from constraint (19)-(20), respectively

υ_o, $\pi_{t,k,m_n,n}$ Dual values from constraint (21)-(22), respectively

$H(i)$, $T(i)$ Head and tail node of arc *i* respectively

η_n, λ_n Source and sink node corresponding to process step *n*, respectively

ω Set of head nodes in the network

ψ Set of tail nodes in the network

φ Set of all the nodes in the network

$\ell_{o,n}$ Set of head nodes of the arcs that correspond to batches with processing start time less than the ready time of the order during process step *n*

$C_{i,j}$ $= (((w_o / s_o)t) - \pi_{t,k,m_n,n}) - (\sigma_{t,m_n,n} / s_o)$, cost assigned to arc (*i*, *j*)

$Y_{i,j}$ Variable = 1 if an arc with tail node *i* and head node *j* is selected; otherwise, = 0

<u>*Subproblem (SP)*</u>

Minimize $$\sum_{i \in \psi} \sum_{j \in \omega} C_{i,j} Y_{i,j} - \upsilon_o - \partial \tag{24}$$

Subject to: $$\sum_{j \in H(\eta_n)} Y_{\eta_n,j} = 1 \tag{25}$$

$$\sum_{j \in T(\lambda_n)} Y_{j,\lambda_n} = 1 \tag{26}$$

$$\sum_{i \in T(j)} Y_{i,j} - \sum_{k \in H(j)} Y_{j,k} = 0 \qquad \forall j \in \varphi - \{\eta_n, \lambda_n\} \tag{27}$$

$$\sum_{i \in T(j)} \sum_{j \in \ell_{o,n}} Y_{i,j} = 0 \tag{28}$$

$$Y_{i,j} \in \{0,1\} \qquad \forall i, j \tag{29}$$

The network structure developed in Section 2.1 is used in SP. Constraint set (25) requires an outgoing arc from the source node be selected, while constraint set (26) requires the selection of an incoming arc into the sink node. Constraint set (27) maintains flow continuity by balancing the incoming and outgoing flow at all nodes in the network other than the source and sink. Finally, as 1) process starting time for batches is fixed and 2) order ready times at process steps are known, an order cannot be assigned to a batch that starts before the order is ready at a given step (28).

As an order's ready time during a process step depends on its completion time in the immediate predecessor step, the SP is solved once for each process step. Hence, during every iteration of the MP, the subproblem is solved the number of times that is equal to the product of the number of orders and the number of process steps. The assumption that the batches are processed on the machines in increasing order of their index also avoids non-linearity in the formulation and makes a

difference only when the orders have non-zero ready times. As the start time of processing the batches is fixed, it may introduce some unnecessary delay in processing a batch. Hence, a post-process step is applied after CG terminates. This post process step adjusts the processing start times of the batches by identifying and eliminating any unnecessary delays and not violating any constraints. In this CG approach, multiple subproblems are solved to generate the complete information required to create a new column and add to the MP rather generating the whole column information in one subproblem. This may be resulting in a degeneracy situation in some instances, where no more improving new columns are created but still some of the columns have negative reduced costs. In such situations, the CG is allowed to run for ten more additional iterations, where the objective value of the restricted MP does not improve, before terminating.

3. Experimental Results

A mini-fab model that represents a complex job shop environment with the characteristics of multiple tool groups, batching, parallel machines, and re-entrant flow is used for our experimentation purposes [2]. Arc labels in Figure 2 indicate step number/process time/batch capacity (in FOUPs).

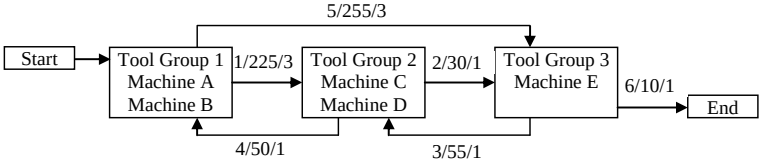


Figure 2. The mini-fab model [2]

The performance of CG-based heuristic for the complex job shop problem is evaluated using the experimental design in Table 1. Many hundreds of integrated circuits (ICs) can be fabricated on a single 300-mm wafer. All experimental parameters are modeled to mimic realistic values in practice. Number of orders is varied at three levels and FOUP capacity is varied at two levels. Order sizes, weights, and ready times are generated from discrete uniform distributions, with each being varied at two levels. For each combination of experimental factors, 10 replications are performed.

Table 1. Experimental Design

Factor	**Level Description**	**# Levels**
Number of orders	10, 15, 20	3
Order size (s_o)	DU$[\upsilon-(\upsilon+1)/2, \upsilon+(\upsilon+1)/2]$, $\upsilon \in \{3,5\}$	2
FOUP capacity (*K*)	$12\beta+1, \beta \in \{1,2\}$	2
Weights (w_o)	1, DU[1, 15]	2
Ready times (r_o)	0, DU[1, 300]	2
Replications per level combination | Total number of instances		10 | 480

CG-based heuristic solutions are compared with the optimization model's solution after a maximum of six hours of computation time ("MIP heuristic"). CPLEX 10.0 is used for solving the optimization models and MIPs in the CG-based heuristics on a 3.40 GHz PC with 2 GB RAM. Computational results are reported in Table 2 in terms of performance ratio (PR), which is defined as the ratio of CG-based heuristic solution to the MIP heuristic solution. Results are given as *a*|*b* denoting instances *without*|*with* order ready times present. It can be observed that the CG solution is close to the MIP heuristic solution for problem instances without ready times. For problem instances with ready times of the orders, CG obtains solutions that are within 11% of the MIP heuristic, on average. In the problem instances with higher number of orders and order weights, PR decreases. This may be because the MIP could not find the optimal or near optimal solution within six hours of computation time for these large problem instances. The computation time limit terminates branch-and-cut before memory limits are exceeded, especially in problems with many orders. This translates into better average performance ratios in 20 order problem instances.

Table 2. Average PR for $FJc|moj(lot), r_o, recrc|\sum w_o C_o$ problem instances

FOUP Size	Order Size	Order Weight	Average Performance Ratios 10 Orders $r_o = 0 \mid r_o > 0$	15 Orders $r_o = 0 \mid r_o > 0$	20 Orders $r_o = 0 \mid r_o > 0$
13	DU[1,5]	1	1.010 | 1.178	1.017 | 1.088	1.020 | 1.112
		DU[1,15]	0.988 | 1.170	0.978 | 1.123	0.958 | 1.049
25	DU[1,5]	1	1.000 | 1.143	0.997 | 1.132	1.003 | 1.092
		DU[1,15]	1.000 | 1.132	0.998 | 1.138	0.973 | 1.051
13	DU[2,8]	1	1.039 | 1.079	1.073 | 1.070	1.017 | 0.970
		DU[1,15]	0.991 | 1.026	1.003 | 1.062	0.931 | 1.030
25	DU[2,8]	1	1.003 | 1.182	1.015 | 1.104	1.020 | 1.121
		DU[1,15]	0.991 | 1.187	0.982 | 1.134	0.978 | 1.087
Average			1.003 | 1.137	1.008 | 1.106	0.988 | 1.064

In terms of computation time, a 20-order problem with $r_o > 0$ and *K*=25 required the largest average computation time of 133 seconds. As the number of orders and FOUP capacity increase, this results in more constraints and decision variables in the CG's MP and SP, which results in increased computation time.

4. Conclusions and Future Work

In this paper, a network-based MIP formulation and a CG-based heuristic approach are presented for the *moj*-CJSSP. A mini-fab model and a representative range of problem instances are selected for experimentation to analyze the effectiveness of the CG approach compared to a time-limited MIP heuristic. From computational results, it can be observed that the CG approach obtains solutions that are very close to the MIP heuristic in the problem instances without ready times. But in the problem instances with ready times, CG obtains solutions within 11% of the MIP heuristic solution. Future research will investigate *moj*(*item*) processing wherein FOUP processing time is dependent on its contents. Lower bounds estimates for minimizing total weighted order completion time in *moj*-CJSSPs will also help to further assess heuristic solution quality.

References

[1] G.B. Dantzig, P. Wolfe (1960), Decomposition principle for linear programs, *Ops Rsch*, **8**(1), 101 – 111.

[2] M.K. Ed Adl, A.A. Rodriguez, K.S. Tsakalis (1996), Hierarchical modeling and control of Reentrant semiconductor manufacturing facilities, *Proc 35th Conf Decision and Control*, Japan, 1736 – 1742.

[3] Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. (1979), Optimization and approximation in deterministic sequencing and scheduling theory: a survey, *Annals of Discrete Math* **5**, 287 – 326.

[4] J. Jampani, S.J. Mason (to appear), Column generation heuristics for multiple machines, multiple orders per job scheduling problems, *Annals of Operations Research*.

[5] S.J. Mason, J.W. Fowler, W.M. Carlyle (2002), A modified shifting bottleneck heuristic for minimizing the total weighted tardiness, *Journal of Scheduling* **5**(3), 247 – 262.

[6] S.J. Mason, K. Oey (2003), Scheduling complex job shops using disjunctive graphs: A cycle elimination procedure, *Int. Journal of Productions Research* **41**(5), 981 – 994.

[7] S.J. Mason, J.W. Fowler, W.M. Carlyle, D.C. Montgomery (2005), Heuristics for minimizing total weighted tardiness in complex job shops, *Int. Journal of Production Research* **43**(10), 1943 – 1963.

[8] L. Moench, R. Driessel (2005), A distributed shifting bottleneck heuristic for complex job shops, *Computers and Industrial Engineering* **49**(3), 363 – 380.

[9] M.I. Ovacik, R. Uzsoy (1994), Exploiting shop floor status information to schedule, *Journal of Manufacturing Systems* **13**(2), 73 – 84.
[10] P. Qu, S.J. Mason (2005), Metaheuristic scheduling of 300mm jobs containing multiple orders, *IEEE Trans on Semiconductor Manufacturing* **18**(4), 633 – 643.
[11] R. Uzsoy, C.Y. Lee, L.A. Martin-Vega (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**(4), 47 – 60.

New Lower Bound for the Bin Packing Problem

Bassem Jarboui
1 FSEGS, route de l'aéroport km 4 Sfax 3018, Tunisie, bassem_jarboui@yahoo.fr
Abdelwaheb Rebai
ISAS, route de l'aéroport km 4 B.P.N° 1013 Sfax 3018, Tunisie, abdelwaheb.rebai@fsegs.rnu.tn

In this paper, we present new classes of lower bound for the one-dimensional bin packing problem, we prove that the worst-case asymptotic performance ratio of this bound is 3/4. Extensive numerical experiments show that this new bound outperforms the best lower bounds from the literature.

Keywords: bin packing, lower bound.

1. Introduction

The bin packing problem can be defined as follows: Given a set L of n items i, with sizes w_i, for any ($1 \leq i \leq n$) and a set of identical bins with capacity C, with an objective to determine the minimal number of bins necessary to pack all the items, so that the sum of the volumes of the items in the same bin does not exceed the capacity of the bin. The problem has many real-world applications (for example in cutting stock or when loading trucks subject to weight limitations). This problem is known to be NP-hard in the strong sense see Garey and Johnson [11]. Consequently, various researches on this problem have mainly focused on the development of polynomial time approximation algorithms. For an exhaustive survey of this area, the reader might refer to Coffman et al [4]. Only a few papers have been devoted to the development of exact algorithms. This work includes the branch and bound algorithm of Martello and Toth [12] and Scholl et al. [14]. In order to get a high performance of such algorithm, it is desirable to have fast lower bounds. There exist several lower bounds for the BPP.

Two types of lower bounds can be developed for the BPP: constructive and destructive bounds. A simple constructive bound is the sum of size of items divided by the size of bin; this bound is called LB_1. This approach also includes the lower bounds (LB_2 and LB_3) proposed by Martello and Toth [12] and the bound based on the Dual Feasible Functions developed by Fekete and Schepers [9]. Destructive bounds are derived in the following way. Starting with a hypothetical lower bound LB, an attempt is made to obtain contradictions to feasibility. If the method provides that there isn't a feasible solution with a number of bins smaller than or equal to LB, then LB+1 is a valid lower bound value. This approach was used by Alvim et al. [1] and Haouari and Gharbi [10].

The goal and the main contribution of this paper is to develop new constructive bound for the BPP. The performances of the new bound are analyzed, both analytically and computationally.

The following part of paper will focus on a review of the best existing lower bounds for the BPP in Section 2. In Section 3, we introduce a new class of bounds; we prove that the worst-case asymptotic performance ratio of the proposed bound is 3/4. In section 4, we give the computational results which illustrate the effectiveness of the proposed bound. Finally, Section 5 contains some concluding remarks.

2. Lower bounds

In this section we recall a review of some well-known linear-time lower bounds for the BPP.

Definition 2.1. Let L be an instance of the problem. LB a lower bound for a minimisation problem, and $OPT(L)$ denotes the optimum value of L. The absolute worst-case ratio and asymptotic worst-case ratio of $LB(L)$ are respectively defined by:

$$r(LB) = \sup\left\{\frac{LB(L)}{OPT(L)} \mid \forall L\right\},\ r_\infty(LB) = \lim_{s\to\infty} \sup\left\{\frac{LB(L)}{OPT(L)} \mid \forall L \text{ with } OPT(L)\geq s\right\}.$$

Let $L=(x_1,\ldots..,x_n)$ be a BPP instance. A simple bound for the BPP is $LB_1(L)$, this bound is obtained by relaxing the constraint witch shows that the items are indivisible:

$$LB_1(L)=\left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$$

The bound $LB_1(L)$ can be computed in $O(n)$ time and it has a worst case performance ratio of 1/2.
A better bound was introduced by Martello and Toth [12] by partitioning the set of items into three subsets $L_1(\alpha)$, $L_2(\alpha)$ and $L_3(\alpha)$ based on the parameter $\alpha \in [0, C/2]$, as follows .

$$L_1(\alpha)=\{j \in L; w_j > C-\alpha\}$$

$$L_2(\alpha)=\{j \in L; C-\alpha \geq w_j > c/2\}$$

$$L_3(\alpha)=\{j \in L; C/2 \geq w_j \geq \alpha\}$$

$LB^{MT}(\alpha ,L)$ presents the number of bins necessary to pack the three subsets of items:

$$LB^{MT}(\alpha,L)=|L_1(\alpha)|+|L_2(\alpha)|+\max\left\{0,\left\lceil \frac{\sum_{j\in L_3(\alpha)} w_j-\left(|L_2(\alpha)|.C-\sum_{j\in L_2(\alpha)} w_j\right)}{C}\right\rceil\right\}$$

The bound $LB_2(L)$ is obtained from the maximal value of $LB^{MT}(\alpha ,L)$ for $\alpha \in [0, C/2]$.

$$LB_2(L)=\max_{\alpha\in[0,C/2]} LB^{MT}(\alpha,L)$$

Martello and Toth [12] have proved that this bound has an asymptotic worst-case ratio 2 /3 and it can be computed in $O(n)$ time if the items are sorted by non-increasing volumes.
Fekete and Schepers [9] propose a new approach based on dual feasible function, to obtaining a new lower bound for bin packing problem. A function u: [0,1]→[0,1] is called dual feasible, if for any finite set S of non-negative real numbers, we have the relation $\sum_{x\in S} x \leq 1 \Rightarrow \sum_{x\in S} u(x) \leq 1$.
Let u be a dual feasible function and let $L=(x_1,\ldots..,x_n)$ be a BPP instance. The lower bound obtained from the transformed BPP instance $u(L)=(u(x_1),\ldots..,u(x_n))$ is also a lower bound for L.
The new lower bounds $LB_*^{(p)}(L)$ can be obtained by applying the same lower bound on the transformed BPP instance $u^{(k)}(L)$ where :

$$LB_*^{(p)}(L)=\max\left\{LB_2(L), \max_{k=2,\ldots,p} LB_2^{(k)}(L)\right\}$$

$$u^{(k)}(x_1)=\begin{cases} x & for\ x(k+1)\in \square \\ \lfloor (k+1)x \rfloor \frac{1}{k} & otherwise \end{cases}$$

$$LB_2^{(k)}(L)=LB_2(u^{(k)}(L))$$

Fekete and Schepers [9] have proved that $LB_*^{(p)}(L)$ has an asymptotic worst-case performance of 3/4 and can be computed in $O(n)$ time if the items are sorted by non-increasing volumes.
Alvim et al. [1] propose the bound $LB_\vartheta(L)$ inspired from the work of Dell'Amico and Martello [7]. This bound can be presented as follows:
Let $\theta=\max_{q=1,\ldots..,n}\left\{q;\ \sum_{i=n-q+1}^{n} w_i \leq C\right\}$
be an upper bound to the number of items per bin for any feasible solution with LB bins.

Theorem 2.1.

$$\vartheta=\max\left\{\max_{\sigma=1,\ldots..,\lfloor n/LB\rfloor}\left\{\sigma:\left\lceil \sum_{i=\sigma}^{n} \frac{w_i}{LB-1}\right\rceil > C\right\},\ \max_{\sigma=1,\ldots..,\lfloor n/LB\rfloor}\left\{\sigma:\sum_{i=1}^{\sigma} w_i \leq C\right\}\right\}$$

is a lower bound to the number of items per bin for any feasible solution with LB bins.

Theorem 2.2. Let LB be a lower bound to the number of bins.

$$LB_{\vartheta}(L)\begin{cases} LB+1, & if\ \ (a),(b),or\,(c)\,is\ verified \\ (a) & \theta < \lceil n/LB \rceil \\ (b) & \left\lceil \sum_{i=LB_{\vartheta}.\vartheta+1}^{n} \frac{w_i}{LB-LB_{\vartheta}} \right\rceil > C \\ (c) & \theta = \vartheta+1\ et\ C_{\vartheta}(LB) > C; \\ LB, & otherwise \end{cases}$$

LB_{ϑ} is an improved lower bound to BPP, where

$$C_{\vartheta}(LB) = \max\left\{ \left\lceil \sum_{i=n-\vartheta.LB_{\vartheta}+1}^{n} w_i / LB_{\vartheta} \right\rceil, \left\lceil \sum_{i=n-\theta.LB_{\theta}+1}^{n} w_i / LB_{\theta} \right\rceil \right\} \text{ and } LB_{\vartheta} = \max\left\{ LB-(n-\vartheta.LB), 0 \right\}$$

3. The new lower bound

In this section we present a new lower bound called $LB_2^{\theta}(L)$

Proposition 3.3. Let *L* be an instance of the *BP* problem and $1 \le k \le \theta$, $k \in \mathbb{N}$, then $\psi(k)$ is an upper bound to the number of bins containing a number of items larger than or equal to *k*.

$$\psi(k) = \min\left\{ \left\lfloor \frac{n}{k} \right\rfloor ; \max_{q=0,\ldots,n-k}\left\{ q; \sum_{i=n-q-k+1}^{n-q} w_i \le C \right\} \right\}$$

Proof. If $\sum_{i=n-q-k+1}^{n-q} w_i > C$, then any item with a size larger than or equal to w_{n-q} cannot be packed into a bin containing *k* items only if this bin contains at least one item with a size smaller than w_{n-q}.

Thus, for any feasible solution of *L*

$$\max_{q=0,\ldots,n}\left\{ q; \sum_{i=n-q-k+1}^{n-q} w_i \le C \right\}$$

is an upper bound to the number of bins containing a number of items larger than or equal to *k*.

Let $k_i \ge k$, the maximal number of bins having a number of items larger than or equal to *k* is bounded by $\max_{q=0,\ldots,n}\left\{ q; \sum_{i=1}^{k} k_i \le n \right\} \le \max_{q=0,\ldots,n}\left\{ q; \sum_{i=1}^{k} k \le n \right\} = \left\lfloor \frac{n}{k} \right\rfloor$

Then,

$$\psi(k) = \min\left\{ \left\lfloor \frac{n}{k} \right\rfloor ; \max_{q=0,\ldots,n-k}\left\{ q; \sum_{i=n-q-k+1}^{n-q} w_i \le C \right\} \right\}$$

Theorem 3.4. Let *L* be an instance of the BP problem and θ the maximal number of items per bin, then

$LB^{\theta}(L) = \psi(k) + \left\lceil \frac{n-\theta.\psi(k)}{\theta-1} \right\rceil$ is a lower bound to the optimum value of *L*.

Proof. Let *m* be the number of bins containing a number of items equal to θ in the optimal solution of *L* and *m'* the remaining bins of this solution.

We have $m' \ge \left\lceil \frac{n-\theta.m}{\theta-1} \right\rceil$.

If $\psi(k) = m$ then $m + m' \le \psi(k) + \left\lceil \frac{n - \theta.\psi(k)}{\theta - 1} \right\rceil$

If $\psi(k) > m$

Let

$$\psi(k) + \frac{n - \theta.\psi(k)}{\theta - 1} = \frac{\theta.\psi(k) - \psi(k) + n - \theta.\psi(k)}{\theta - 1}$$

$$= \frac{n - \psi(k)}{\theta - 1} < \frac{n - m}{\theta - 1} = m + \frac{n - \theta.m}{\theta - 1} \le m + \left\lceil \frac{n - \theta.m}{\theta - 1} \right\rceil$$

We have $\psi(\theta) + \frac{n - \theta.\psi(k)}{\theta - 1} < m + m'$ then $\psi(k) + \left\lceil \frac{n - \theta.\psi(k)}{\theta - 1} \right\rceil \le m + m'$

Proposition 3.5. The computation of $LB^{\theta}(L)$ requires $O(n)$ time for pre-sorted items.

Proof. Let $q^* = \max_{q=1,\ldots,n-\theta+1} \left\{ q; \sum_{i=n-q-\theta+2}^{n-q+1} w_i \le C \right\}$

For any value of θ, it is easy to say that if θ is valid then $q^* \ge 1$, therefore to compute q^*, we can start from q=2. We have $\sum_{i=n-q-\theta+1}^{n-q} w_i = \sum_{i=n-q-\theta+2}^{n-q+1} w_i + w_{n-q-\theta+1} - w_{n-q+1}$, these imply that the computation of q^* requires a constant number multiplied by q^*. We have $\theta \le n$ and $q^* \le n$, which makes the computation of q^* require $O(n)$ time for computing θ and $O(n)$ time for computing q^*. Then $\psi(\theta)$ is computed in $O(n)$ time.

So, $LB^{\theta}(L) = \psi(\theta) + \left\lceil \frac{n - \theta.\psi(\theta)}{\theta - 1} \right\rceil$ can be computed in $O(n)$ time.

In what follows, we present a refinement of the above lower bound $LB^{\theta}(L)$. Similar to the bound $LB_2(L)$, our bound is based on a partition of the set of items into two subsets $L_1(\alpha)$ and $L_4(\alpha)$, where

$L_1(\alpha) = \{ j \in L; w_j > C - \alpha \}$

$L_4(\alpha) = \{ j \in L; C - \alpha \ge w_j > \alpha \}$

We define $LB^{\theta}_{MT}(\alpha, L)$ by

$LB^{\theta}_{MT}(\alpha, L) = |L_1(\alpha)| + \max \{ LB_1(L_4(\alpha)), LB^{\theta}(L_4(\alpha)) \}$

Then a valid lower bound is

$LB^{\theta}_2(L) = \max_{\alpha \in [0, C/2]} LB^{\theta}_{MT}(\alpha, L)$

Proposition 3.6. The computation of $LB^{\theta}_2(L)$ requires $O(n)$ time for pre-sorted items.

Proof. Martello and Toth[12] have proved that all the updating of $LB_1(L_4)$ can be computed in $O(n)$ time, then it suffices to prove that all the updating of $LB^{\theta}(L)$ can be computed in $O(n)$ time. To compute $LB^{\theta}_2(L, \alpha)$, we consider the α values which are limited by $\alpha_i = \{ w_i \in L : w_i \le C/2 \}$.

Let $\theta(\alpha_i)$ be the value of the maximal number of items per bin associated to α_i, θ_l the set of values of the maximal number of items per bin obtained during the computation of $LB^{\theta}_2(L, \alpha)$ with $l = 1, \ldots, r$ and $\theta_l \ge \ldots \ge \theta_l \ge \theta_{l+1} \ge \ldots \ge \theta_r$.

Let n_l be the number of values taken by α_i decremented by 1, obtained directly after the passage from θ_{l+1} to θ_l

and $q^*_l = \max_{q=0,\ldots,n-n_l-\theta_l} \left\{ q; \sum_{j=n-n_l-q-\theta_l+1}^{n-n_l-q} w_j \le C \right\}$

Computation of all the updating of maximal number of items per bin associated to α_i

Let $\theta(\alpha_i) = \theta_l$,

If $i - n_l \le q_l^*$

then $\theta(\alpha_i) = \theta(\alpha_{i-1})$

otherwise we have to compute the new value of $\theta(\alpha_i)$

if $q_l^* \ge \theta_l$,

the computation of θ_{l+1} would begin from the item with index $\sum_{i=1}^{l} q_i^*$ then the total number of operations is equal to $\sum_{l=1}^{r} \theta_l \le \sum_{l=1}^{r} q_l^*$, it is clear that $\sum_{l=1}^{r} q_l^* \le n$, we then have $\sum_{l=1}^{r} \theta_l \le n$

if $q_l^* < \theta_l$ and $\theta_{l+1} > \theta_l - q_l^*$,

then the computation of the value θ_{l+1} requires a subtraction of the items going from $w_{n-n_l-q_l^*+1}$ through w_{n-n_l} and the addition of items which goes from $w_{n-n_l-\theta_{l+1}-q_l^*+1}$ through $w_{n-n_l-\theta_l}$ thus, the total number of operations will be equal to :

$$\theta_1 + \sum_{l=1}^{r-1}(n-n_l) - (n-n_l-q_l^*+1) + (n-n_l-\theta_l) - (n-n_l-\theta_{l+1}-q_l^*+1)$$

$$= \theta_1 + \sum_{l=1}^{r-1}(2q_l^* + \theta_{l+1} - \theta_l - 2) = \theta_1 + \sum_{l=1}^{r-1} 2q_l^* - 2(r-1) \le 3n - 2(r-1)$$

if $q_l^* < \theta_l$ and $\theta_{l+1} = \theta_l - q_l^*$,

then the computation of the value θ_{l+1} requires a subtraction of the items going from $w_{n-n_l-q_l^*+1}$ to w_{n-n_l} and the addition of items which goes from $w_{n-n_l-\theta_{l+1}-q_l^*+1}$ through $w_{n-n_l-\theta_l}$.This implies that the total number of operations will be equal to :

$$\theta_1 + \sum_{l=1}^{r-1}\left[(n-n_l) - (n-n_l-q_l^*+1)\right] = \theta_1 + \sum_{l=1}^{r-1} q_l^* - (r-1) \le 2n - (r-1)$$

Computation of maximal number of bins for any value of α_i

We have

$$\psi(\theta(\alpha_i)) = \min\left\{\left\lfloor \frac{|L_4|}{\theta(\alpha_i)} \right\rfloor; \max_{q=0,\ldots,n-i-\theta(\alpha_i)} \left\{ q; \sum_{i=n-i-q-\theta(\alpha_i)+1}^{n-i-q} w_i \le C \right\}\right\} \text{ And } \theta(\alpha_i) = \theta_l$$

If $\theta(\alpha_i) = \theta(\alpha_{i-1})$

then $\max_{q=0,\ldots,n-i-\theta(\alpha_i)} \left\{ q; \sum_{i=n-j-q-\theta(\alpha_i)+1}^{n-i-q} w_j \le C \right\} = q_l^* - (i - n_l)$

this implies that, $\psi(\theta(\alpha_i))$ can be computed in linear time.

if $\theta(\alpha_i) < \theta(\alpha_{i-1})$,

then, to compute q_l^*, we consider the set of items used for computing θ_l and we start with $q_l^* = 2$. We have proved in Proposition 3.3 that the computation of q_l^* requires a constant number multiplied by q_l^*, we have $\sum_{l=1}^{r} q_l^* \le n$, these imply that $\psi(\theta)$ can be computed in a linear time for all the updating. Then, the computation of $LB_2^{\theta}(L)$ requires $O(n)$ time for pre-sorted items.

Theorem 3.7. Let L be an instance of the BP problem with the size of items larger than $C/3$. Then, $LB_2^{\theta}(L) = OPT(L)$

Proof.

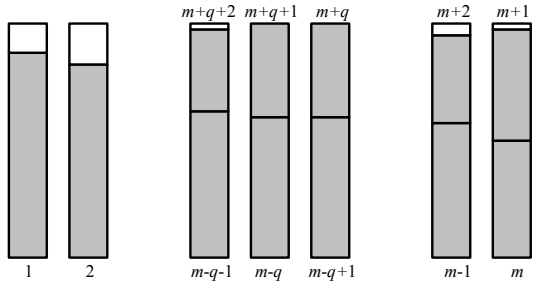


Figure 1: Normal form of an optimal solution for a BPP instance with sizes $w_i > C/3$

Let $m = OPT(L)$ be the number of bins of optimal solution and $\theta = 2$ the maximal number of items per bin.

If $m = \frac{n}{2}$,

then for any value of $\alpha = 0$, we have

$$LB_2^{\theta}(L) \geq |L_1| + \left\lceil \frac{|L_4|}{\theta} \right\rceil = 0 + \frac{n}{2} = m = OPT(L).$$

If $m > \frac{n}{2}$,

then, there exists at least one item w_{m+q+1} which cannot be packed with any other item having a size larger than $w_{m-q} > C/2$ for any feasible solution with m bins. Then, for any value of $\alpha = C - w_{m-q}$, we have

$$LB_2^{\theta} \geq |L_1| + \left\lceil \frac{|L_4|}{\theta} \right\rceil \geq m - q - 1 + \frac{((m+q+1)-(m-q)+1)}{2} = m = OPT(L)$$

Then, LB_2^{θ} is optimal for any solution with the items' sizes larger than $C/3$

Theorem 3.8.

$$r_{\infty}(LB_2^{\theta}) = \frac{3}{4}$$

Proof. Let $L' \in L$ be the sub-set of items with a size $w_i > C/k$. Crainic and al.[5] have proved that for a given value of $k > 1, k \in \mathbb{N}$ and for a lower bound LB of the BPP, if $LB(L') = OPT(L')$, $LB(L) \geq LB(L')$ and $LB(L) \geq LB_1(L)$, the asymptotic worst-case ratio of LB is

$$r_{\infty}(LB) = \frac{k}{k+1}.$$

Then, it suffices to prove that $LB_2^{\theta}(L) \geq LB_2^{\theta}(L')$.

We have proved in Theorem 3.7 that the optimal solution of any instance with the size of items larger than $C/3$ can be reached from the value of $\alpha = C - w_{m-q} > C/3$.

For any value of $\alpha > C/3$, we have $LB_{MT}^{\theta}(\alpha, L) = LB_{MT}^{\theta}(\alpha, L')$ then,

$LB_2^{\theta}(L) \geq OPT(L') = LB_2^{\theta}(L')$

Thus, $r_{\infty}(LB_2^{\theta}) = \frac{3}{4}$.

4. Computational results

A computational experiment was conducted to evaluate the performance of the proposed new lower bounds. The algorithms were coded in C++, and the computational experiments were run on a PC Pentium IV, 3.2 GHz Personal Computer with 512 MB RAM.

The class of problems is generated in a similar way to the instances described by Fekete and Schepers [9]. For a given number n of items, the sizes were generated randomly with uniform distribution on the sets S1 ={1,…,100}, S2 ={1,…,90}, S3 ={1,…,80} , S4 ={20,…,80} , S5 ={20,…,70}, for the container size C=100.

We consider instances with 32, 100, 316, and 1000 items. One thousand random instances are generated for each problem class and number of items.

We measure the quality of bounds according to the number of improved instances and the total gaps in the number of bins. Let "imp" and "gap" be respectively the number of instances improved by *LB* from *LB'* (*LB*/*LB'*) and the total gaps obtained by *LB*-*LB'*.

We define a new bound by exploiting the dual function proposed by Fekete and Schepers [9] which is as follows:

$$LB_{FS}^{\theta}(L,p) = \max\left\{LB_2^{\theta}(L), \max_{k\in[0,p]}\left\{LB_2^{\theta}(u^{(k)})\right\}\right\}.$$

Table 1: number of improvements and total gaps in the number of bins for 1000 instances

		$LB_2^{\theta}(L)/LB_2(L)$		$LB_{FS}^{\theta}(L,20)/LB_*^{(20)}(L)$		$LB_{FS}^{\theta}(L,50)/LB_*^{(50)}(L)$		$LB_{FS}^{\theta}(L,100)/LB_*^{(100)}(L)$	
		imp	gap	imp	Gap	imp	gap	imp	gap
1 100	32	90	90	169	169	169	169	167	167
	100	53	54	111	112	101	102	99	100
	316	70	86	151	176	112	141	110	139
	1000	183	243	356	755	109	156	126	173
1 90	32	96	96	192	192	189	189	185	185
	100	59	64	115	115	100	101	99	100
	316	54	68	155	187	114	149	117	152
	1000	121	184	294	612	113	192	115	194
1 80	32	65	65	153	153	151	151	151	151
	100	33	39	53	53	68	69	66	67
	316	9	18	19	23	35	47	36	48
	1000	0	0	0	0	19	22	18	21
20 80	32	67	67	153	153	158	158	154	154
	100	54	59	116	117	108	115	105	112
	316	113	147	252	370	146	199	151	204
	1000	221	373	492	1437	104	159	114	169
20 70	32	54	54	67	67	73	74	73	74
	100	12	16	14	14	20	22	19	21
	316	1	1	1	2	1	3	2	4
	1000	0	0	0	0	0	0	0	0

Table 1 shows the results obtained by applying the bounds $LB_2^{\theta}(L)$, $LB_{FS}^{\theta}(L,20)$, $LB_{FS}^{\theta}(L,50)$ and $LB_{FS}^{\theta}(L,100)$ on the class of problems. We note that these bounds outperform respectively the bounds $LB_2(L)$, $LB_*^{(20)}(L)$, $LB_*^{(50)}(L)$ and $LB_*^{(100)}(L)$. Table 2 reports the average CPU time (in 10^{-3} s) necessary to compute the different bounds. We note that $LB_{\varphi}(L)$ is a very fast bound which provides high quality bounds under an acceptable computational effort.

Table 2. Average CPU time (in 10^{-3} s)

	LB_2	$LB_*^{(20)}$	$LB_*^{(50)}$	$LB_*^{(100)}$	$LB_2^{\theta}(L)$	$LB_{FS}^{\theta}(20)$	$LB_{FS}^{\theta}(50)$	$LB_{FS}^{\theta}(100)$
32	0.0231	0.761	1,3282	4.3246	0,0468	1,2134	3,3158	6,7622
100	0,0372	1,8054	3,6428	9,4381	0,0776	2,9182	7,7444	17,3302
316	0,1412	4,9798	12,1334	25,9674	0,2564	9,1594	26,1022	50,553
1000	0,3941	20,5464	37,9156	123,495	0,7016	30,2448	77,0992	138,879

5. Conclusion

A new lower bound for the bin packing problem is presented. We have proved that this bound has an asymptotic worst case ratio of 3/4. The computational results have proved the effectiveness of this proposed bound to obtain the best results among existing lower bounds. The new bounds are easy to be implemented and often provide high quality bounds under an acceptable computational effort.

References

[1] A. Alvim, C. Ribeiro, F. Glover, D. Aloise (2004), A hybrid improvement heuristic for the one-dimensional bin packing problem, *Journal of Heuristics* **10**, 205 – 229.

[2] J.M. Bourjolly, V. Rebetez (2005), An analysis of lower bound procedures, *Computers and Operations Research* **32**, 395 – 405.

[3] B. Chen, B. Srivastava (1996), An improved lower bound for the bin packing problem, *Discrete Applied Mathematics* **66**, 81 – 94.

[4] E.G. Coffman Jr., M.R. Gary, D.S. Johnson (1997), *Approximation algorithms for bin packing*: *a survey*, *in*: *D. Hochbaum* (*Ed.*), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing, Boston.

[5] G. Crainic, G. Perboli, M. Pezzuto, R. Tadei R, Computing the asymptotic worst-case of bin packing lower bounds. *European Journal of Operational Research*, (article in press).

[6] G. Crainic, G. Perboli, M. Pezzuto, R. Tadei R (2007), New bin packing fast lower bounds. *Computers and Operations Research* **34**, 3439 – 3457.

[7] M. Dell'Amico, S. Martello (1995), Optimal scheduling of tasks on identical parallel processors, *ORSA J. Comput* **7**, 191 – 200.

[8] S. Elhedhli (2005), Ranking lower bounds for the bin packing problem, *European Journal of Operational Research* **160,** 34 – 46.

[9] S. Fekete, J. Schepers (2001), New classes of fast lower bounds for bin packing problems, *Math. Programming* **91**, 11 – 31.

[10] M. Haouaria, A. Gharbia (2005), Fast lifting procedures for the bin packing problem, *Discrete Optimization* **2**, 201 – 218.

[11] M.R. Garey, D.S. Johnson (1979), *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco.

[12] S. Martello, P. Toth (1990), *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, Chichester.

[13] S. Martello, P. Toth (1990), Lower bounds and reduction procedures for the bin packing problem, *Discrete Applied Mathematics* **28**, 59 – 70.

[14] A. Scholl, R. Klein, C. Jürgens (1997), BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem, *Computers and Operations Research* **24**, 627–645.

[15] F. Vanderbeck (1999), Computational study of a column generation algorithm for bin-packing and cutting stock problems, *Mathematical Programming* **86**, 565 – 594.

Optimization vs. Persistence in Scheduling: Heuristics to Manage Uncertainty in On-Demand Air Travel

Itir Z. Karaesmen
University of Maryland, College Park, MD, USA, ikaraes@rhsmith.umd.edu
Wei Yang
Long Island University, Long Island, NY, USA wei.yang@liu.edu
Pinar Keskinocak
Georgia Institute of Technology, Atlanta, GA, USA, pinar@isye.gatech.edu

We study the scheduling problem for fractional management companies (FMCs) that provide on-demand air travel services. FMCs operate in a highly uncertain and dynamic environment where frequent changes in supply (e.g., due to aircraft break-downs) and demand (e.g., trip cancellations, new trip requests) occur throughout a scheduling horizon. Certain logistical factors require FMCs to prepare their schedules in advance without complete knowledge of the aircraft availability and the trips. These advance schedules are later updated as more information becomes available. However, modified schedules are desired to remain both cost-effective and persistent (i.e., close to the original schedule). We propose a reserve-fleet heuristic that pro-actively solves the persistence problem by enforcing idleness of the aircraft in creating the original schedule. We discuss how the reserve-fleet size can be determined, what the composition of the reserve-fleet should be, and how the original schedule is modified using this reserve-fleet idea. The reserve-fleet heuristic takes advantage of the set-partitioning formulations and branch-and-price methods developed earlier for FMCs. We also present results of computational experiments that quantify the persistence vs. optimization trade-off for FMCs.

Keywords: transport scheduling, applications, heuristics, aviation, rescheduling, robustness
History: submitted January 31, 2007; revised June 8, 2007.

1 Introduction

On-demand air travel has seen steady increase over the past decade and is now common among business executives, celebrities, and others seeking the convenience and flexibility of being able to travel any time, anywhere. Charter companies and fractional management companies (FMCs) are two major providers of on-demand air travel. These companies take full responsibility for managing aircraft, crew and other aspects of air travel while their customers enjoy the benefits of private aviation. While charter companies work similar to limousine companies with no commitments or availability guarantees, FMCs provide their services through fractional ownership or jet-card programs where the customers are entitled to a certain amount of flight time on the company jets, and the service is guaranteed with as short as 4 hours of notice. On-demand air travel services by FMCs started in the late 1980s with only a handful of fractional-owners and aircraft. The industry forecasts the number of fractional-owners to reach 7000 by the end of 2007 [14], and the largest FMC in the U.S. and Europe, NetJets, currently has a fleet of 665 private jets, making the company one of the largest airlines in the world [15].

Despite the growth in the industry, it is a big challenge for FMCs to remain profitable, given their high fixed and variable costs, such as flight costs, maintenance and repair costs of the aircraft, and costs associated with crew overtime and routing the crew (e.g., from/to their home-base to/from

the location of the aircraft at the start/end of their duty). While the FMC charges a fee to its customers for the actual flight time, the company bears the cost of re-positioning (routing) an aircraft from its location to a customer's point of origin. In fact, re-positioning may constitute as much as 25-35% of the total flight time of the fleet for a FMC [8, 21]. Another cost component for FMCs is subcontracting. A FMC subcontracts a trip to a charter company or to another FMC when there are no aircraft in its fleet to serve a customer's request on time. Netjets reports having spent $200 million on charters in 2005 [8].

A FMC must have efficient operations on a daily basis to manage all these costs. Scheduling of the aircraft and the crew to satisfy customers' requests remain the two most critical operational decisions for FMCs. Several researchers and practitioners have studied different aspects of the scheduling problem for FMCs, including scheduling of the aircraft only [11, 4, 5, 6] and combined aircraft and crew scheduling [16, 21]. Martin et al. [13], Hicks et al. [9], Karaesmen et al. [10] and Yang et al. [20] introduce decision support tools (DST) for FMCs. Several models and methods are proposed in these papers to obtain optimal or near-optimal schedules assuming perfect information about customers' requests, availability of aircraft and the crew. These models play an important role in meeting the FMCs need to prepare an advance schedule, often called a *Master Schedule*, that is required by some airports 24-hours in advance, and also needed for logistical planning of scheduled maintenance activities and crew swaps.

Given that the nature of the business and customer characteristics make it almost impossible to predict the demand in advance, FMCs modify the Master Schedule as new information (e.g., trip cancellations, new trip requests, aircraft break-downs, early completion of scheduled maintenance for aircraft) becomes available [10]. Based on the operations of a FMC, 20% of trips are requested with as little as 4 hours notice, and 30% of the trips requested are changed at least once within 48 hours of flight time [9]. A FMC is faced with uncertainty not only in demand, but also in supply. A total of 49 mid-day and 104 overnight unscheduled maintenance events were recorded during a one-month period for a FMC that has 35 aircraft [21]. It is often desirable that a modified schedule remains close to the Master Schedule while keeping the costs low.

Although rescheduling is of utmost importance for FMCs, so far it has received limited attention in the literature [9, 13]. For a slightly different business model for on-demand air transportation, a local search heuristic is suggested to modify a Master Schedule [6]. Our goal is to find scheduling solutions that achieve a good tradeoff between optimality and *persistence*, i.e., the modified solution should be nearly optimal with respect to standard criteria without deviating too much from the previous solution. Rather than penalizing deviations from a given schedule (as in [9]), we propose an approach that is pro-active in dealing with schedule persistence: Our heuristic enforces idleness of the aircraft in creating a Master Schedule in order to retain scheduling flexibility to deal with future supply/demand issues.

There are other industries in which deterministic optimization and/or scheduling is carried out despite uncertainty in demand and supply. For example, commercial airlines face supply disruptions due to aircraft or crew unavailability and rely on emergency or recovery tactics (e.g., delaying flight departures, cancelling flights, re-routing aircraft or passengers, re-assigning crews) to get their operations back inline with their original schedules [2, 7, 22]. Traditionally, commercial airlines have first created a schedule in order to minimize costs or maximize aircraft utilization, then tried to retain persistency via recovery efforts. More recently, the concept of *robustness* of airline schedules has received attention; i.e., airlines can strategically plan for disruptions while creating their schedules via robust fleet assignment [18], robust aircraft routing [12] and robust crew assignment [17]. In contrast to commercial airlines, FMCs face both supply and demand disruptions and need to consider optimization and persistence simultaneously in scheduling the aircraft and the crew. On the up side, since FMCs operate on-demand (they can be viewed as

"unscheduled airlines") they have more flexibility in terms of aircraft and crew re-assignments.

Schedule perturbations and obtaining robust schedules are also of concern in manufacturing [1, 19]. The idea of *under-capacity scheduling* [1] is in spirit close to what we are proposing for a FMC. Unfortunately, the models and solution methods developed for production scheduling are not adequate for a FMC because of the differences in the objectives and the operational characteristics.

In this paper, we propose a heuristic to obtain persistent and cost-effective schedules for FMCs. Our approach is *unique* in the way that it aims to create a schedule that is flexible enough to deal with future supply and demand changes. We achieve such scheduling flexibility by enforcing idleness of the aircraft in creating a Master Schedule and call this the *reserve-fleet heuristic*. We discuss how the reserve-fleet size can be determined, what the composition of the reserve-fleet should be, and how the original schedule is modified using the reserve-fleet heuristic. We also report results of computational experiments that quantify the cost-effectiveness vs. persistence of our heuristic. To the best of our knowledge, our work is *the first* that studies the trade-off between optimization and persistence for FMCs. In the remainder of the paper, we first define the scheduling problem in Section 2. Our heuristic is introduced in Section 3, and results of computational experiments are reported in Section 4. We conclude and discuss future research directions in Section 5.

2 Problem Definition

Given a Master Schedule created at time $t = 0$ (Stage-0) for a pre-defined planning horizon, consider the state of the system at time $t > 0$ (Stage-1). Part of the schedule may be activated by that time (i.e., some aircraft may already be serving trips or in the process of repositioning), and we may have new information about trip arrivals, changes or cancellations, as well as changes in aircraft availability. Given the new information, our goal is to create a modified schedule which stays close to the original one (in terms of trip-aircraft pairings) as much as possible while still minimizing repositioning and subcontracting costs.

Throughout the paper, index i denotes aircraft and j represents trips. We denote $N_0 = \{1, ..., n_0\}$ and $M_0 = \{1, ..., m_0\}$ as the set of all available aircraft and requested trips, respectively, at time $t = 0$. We further use the following notation:

Ω_i : the set of feasible routes for aircraft i, $i \in \bar{N}_0$,
p : a feasible route, $p \in \Omega_i$,
c_{ip} : the total cost of taking route p by aircraft i, $i \in \bar{N}_0$ and $p \in \Omega_i$,
a^i_{jp} = 1, if trip j is on route p; 0, otherwise, $p \in \Omega_i$, $i \in \bar{N}_0$ and $j \in M_0$,
θ_{ip} = 1, if aircraft i takes route p; 0, otherwise, $p \in \Omega_i$ and $i \in \bar{N}_0$

where $\bar{N}_0 = \{1, 2, ..., n_0 + m_0\}$ is the set of all available aircraft such that $\{1, ..., n_0\}$ represents the fleet and $\{n_0 + 1, ..., n_0 + m_0\}$ represents charter aircraft for subcontracting the trips at Stage-0. For each $i \in \{n_0 + j : j = 1, ..., m_0\}$, Ω_i consists of a single route that takes only trip j with the route cost being the cost of subcontracting trip j.

In our earlier research, we proposed the following set partitioning formulation (SP) to create the Master Schedule for the combined aircraft and crew scheduling problem, where each aircraft is coupled with its crew [20] (i.e., aircraft availability is constrained not only by maintenance related activities but also allowable duty and rest periods for the crew):

$$(SP) \quad Z_0^{SP} = \min \sum_{i \in \bar{N}_0} \sum_{p \in \Omega_i} c_{ip}\theta_{ip} \tag{1}$$

$$s.t. \quad \sum_{i \in \bar{N}_0} \sum_{p \in \Omega_i} a^i_{jp}\theta_{ip} = 1, \quad j \in M_0 \tag{2}$$

$$\sum_{p \in \Omega_i} \theta_{ip} \leq 1, \qquad i \in \bar{N}_0 \tag{3}$$

$$\theta_{ip} \in \{0,1\}, \qquad i \in \bar{N}_0, \text{ and } p \in \Omega_i. \tag{4}$$

Constraints (2) ensure that each trip is covered and constraints (3) ensure that an aircraft is assigned to at most one route. Note that the problem is always feasible because the trips can be subcontracted. In [20], we used a branch and price method which combines column generation with branch-and-bound to obtain a near-optimal integral solution to SP; we refer to that procedure as BP. Note that the routes for each aircraft can easily be created in this problem by pre-processing the input data to identify the aircraft-trip compatibility and the availability of an aircraft to take a trip or a pair of trips back-to-back (see [10, 20] for pre-processing aircraft- and trip-related information and [13] for pre-processing crew-related information). Our heuristic takes advantage of the SP formulation and the BP method; details are provided below.

3 Reserve-Fleet Heuristic

As demand and supply changes are observed, the Master Schedule needs to be modified to remain feasible and cost-effective. Re-optimizing the schedule by re-solving SP may result in a drastically different output, even with a small change to the input data. This phenomenon is quite common in mathematical programming [3], and techniques for incorporating persistence into optimization models include (i) fixing a subset of variables to preferred values, (ii) converting certain variables into persistent variables that incur linear penalties for deviating from their preferred values (as in Hicks et al. [9]), and (iii) using persistent constraints to ensure that aggregate quantities achieve preferred values or ranges of values and to allow penalized deviations to occur. Notice that all these approaches deal with persistence only at the time of rescheduling (Stage-1). In contrast, we propose a pro-active approach (at Stage-0) to achieve schedule persistence.

We propose to create a Master Schedule at Stage-0 which may be slightly suboptimal in terms of costs but is flexible enough to accommodate future changes without significantly affecting the existing trip-aircraft assignments. The main idea is to use only a fraction of the fleet in creating the Master Schedule and keep the remaining aircraft idle as a *reserve-fleet* to respond to future demand/supply changes. Recall that a typical FMC only knows about 75-80% of the trips while creating its Master Schedule [20], and the reserve-fleet can help reduce the number of future schedule perturbations. While this is a simple idea, there are three key issues: What is the right size for the reserve-fleet? Which aircraft should be in the reserve? How does one adjust/modify a schedule that was created using the reserve-fleet idea? We provide the answers below assuming the FMC owns a homogeneous fleet.

Determining the Reserve-Fleet Size: Let r be the size of the reserve as a fraction of the fleet, i.e., $(1-r)n_0$ is the number of aircraft considered for scheduling at Stage-0. Due to the complexity of the FMC's problem and the need for practical methods, we favor a systematic analysis to determine the value of r. The company can use historical data (or use simulations to generate random data that is representative of company's operations) to see how different values of r would have affected the scheduling performance in the past (or in the simulations). Given demand and supply information, one can re-solve the scheduling problems using the reserve-fleet idea and compute estimates of costs and number of schedule perturbations given a value of r. Experimenting with different values of r enables the decision-makers to choose the reserve-fleet size that provides the best trade-off between optimization and persistence. We illustrate the systematic approach on randomly generated data in Section 4. However, we first need to decide which aircraft goes into the reserve for a given r.

Determining the Composition of the Reserve-Fleet: Suppose r is given. We can solve SP using BP to create a Master Schedule if we know a priori which aircraft is kept for the reserve. This can be done by randomly choosing the aircraft that goes into the reserve. While too simple, this randomization approach may provide reasonable results due to the complexity of the problem. We call this the *randomized-reserve-fleet (RRF)* approach. Another approach is to add $\tilde{m} = \lceil rn_0 \rceil$ (the smallest integer exceeding the reserve fleet size) *dummy trips* to set M_0. These trips can be taken by any of the aircraft in set $\bar{N}_0$, and their repositioning and flight costs are zero. However, they cannot be on the same route with any other trip, and they have extremely high subcontracting costs. In other words, $\tilde{m}$ feasible routes with extremely high costs are added to the set Ω_i for each $i \in \bar{N}_0$. Using these expanded set of trips and routes, we re-formulate SP. The optimal solution to this modified set-partitioning problem assigns $\tilde{m}$ company aircraft to the dummy trips to avoid the high cost of subcontracting. The aircraft assigned to these dummy trips provide the optimal composition for the reserve fleet. We call this approach the *optimized-reserve-fleet (ORF)*. We test the effectiveness of both RRF and ORF using computational experiments.

Rescheduling at Stage-1 Using the Reserve-Fleet: The goal of the reserve fleet heuristic is to give flexibility to schedulers in the future. The new schedule can be computed by re-solving SP with the updated information at Stage-1, by (partially) fixing trip-aircraft assignments from Stage-0 *except for* subcontracted trips, trip cancellations, or aircraft breakdowns. Note that the fixed assignments only change the column generation step at BP. Feasible routes can be obtained easily if the fixed assignments are included in pre-processing in order to determine feasible aircraft-trip assignments [10].

4 Computational Results

The purpose of our computational experiments is to quantify the optimization and persistence trade-off, to determine the appropriate reserve-fleet size based on that trade-off, and to benchmark the performance of the reserve-fleet heuristic. We use randomly generated data and work in a simplified environment. We assume that both Stage-0 and Stage-1 problems are solved before the Master Schedule is activated; this is typical of FMCs. We assume the only uncertainty is on the demand side, and that the entire fleet (of size n_0) is available at both stages. We also assume there are no cancellations. Hence, the only new information at Stage-1 is regarding additional trips. As a comparison to the performance of the reserve-fleet heuristic, we define an approach called *re-optimization* which solves SP at Stage-0 using the entire fleet, and resolves SP with updated information at Stage-1 without fixing any of the prior trip-aircraft assignments. The Stage-1 solution of this approach is the *hindsight optimal* solution that minimizes the total costs. However, re-optimization at Stage-1 is expected to lead to a high number of schedule perturbations.

We use the following parameters: The scheduling horizon is 36 hours. Only 80% of the trips are known at Stage-0. Aircraft locations and trip origins/destinations as well as departure times are generated randomly: We first choose 100 cities that are uniformly distributed on a 100 by 100 grid. The travel time between two adjacent points on the grid is assumed to be 3 minutes, therefore the longest flight is about 7 hours. We randomly select a departure location and a destination location for each trip with flight time greater than 30 minutes. Setting the time point for optimization at zero, we are planning the events between now and 2160 minutes later. The earliest available time of each aircraft is uniformly distributed between 0 and 1200 minutes. The departure time for a trip is uniformly chosen between 0 and 1800 minutes. We denote each experiment by $[r, (1-r)n_0, m_0, n_0, m_1]$ where $(1-r)n_0$ is the number of aircraft used in scheduling at Stage-0 and $m_1 = \frac{m_0}{0.8}$ is the total number of trips at Stage-1. We generate 10 random instances for each

experiment. All instances in an experiment have the same m_0 and m_1, but trips differ in trip departure times and origins/destinations from one instance to another.

We compute two measures in each experiment: (i) The *average optimality gap* of the reserve-fleet heuristic is the percentage difference between the hindsight-optimal costs and the cost of the reserve-fleet solution. (ii) The *average persistence* is the average of the ratio of number of trips that have the same assignment at Stage-0 and Stage-1 (i.e., if a trip j is assigned to an aircraft $i \in N$, then j remains assigned to i at Stage-1, or if j is chartered at Stage-0, it remains chartered at Stage-1) to number of trips known at Stage-0 (m_0); this measure is an indication of the FMC's 'ability to schedule in advance' and is based on the total number of schedule perturbations. We also report the average number of trips subcontracted by re-optimization and reserve-fleet solutions at Stage-1.

Table 1 summarizes the results of our experiments. We implemented both the randomized procedure RRF and optimization-based procedure ORF. We only report results of experiments where $n_0 = 50$ and $m_1 \in \{50, 100, 150\}$. Results of our experiments with higher fleet sizes are omitted because the load factor (relative size of the fleet to the number of trips) impacts our results more than the actual fleet size. Here are the main observations from Table 1: (i) RRF and ORF have similar performance; neither one dominates the other. (ii) The reserve-fleet approach is most appropriate for moderately high load factors ($l = 2$ when $n_0 = 50, m_1 = 100$). This is expected because when the load factor is very low, there are unused aircraft and demand/supply changes can be handled fairly easily with this excess capacity. When the load factor is high, a significant number of trips are chartered. In that case, the reserve fleet further limits the capabilities at Stage-0, leads to a higher number of subcontracted trips. Consequently, the solution in Stage-1 tries to reassign those subcontracted trips to company aircraft, adversely affecting the schedule persistence. (iii) Average persistence decreases as r increases in both RRF and ORF. However, the average optimality gap fluctuates with no apparent pattern. This is expected: First of all, when $r = 1$, the optimality gap of the reserve-fleet heuristics is 0% because Stage-1 solution of reserve-fleet is the same as the hindsight optimal solution in that case. Hence, as $r \to 1$, the optimality gap gets smaller. Second, there is a delicate balance between the inefficiency at Stage-1 due to limited availability of aircraft because of prior assignments, and the inefficiency at Stage-0 due to limited number of aircraft. Depending on the size of the reserve fleet, either source of inefficiency becomes the dominant factor in the optimality gap. The randomness in the data sets make it difficult to detect when Stage-1 or Stage-0 inflexibility is more important, and how the size of the reserve fleet affects that. Note that the differences in the optimality gaps can also be explained by the total number of charter aircraft at Stage-1. The number of charters used also fluctuate with r.

The fleet size can be chosen by detailed examination of the results in Table 1. Different rules need to be developed for different load factors. For instance, when $m_1 = 150$, there is an improvement in optimality gap when r increases from 0 to 10%. However, further increases in r do not yield significant decreases in optimality gap and lead to low schedule persistence. In that respect, r can be 10-20% for high load factors. This choice guarantees 5% gap in costs and 50% schedule persistence. When $m_1 = 100$, $r = 20\%$ guarantees about 68-70% schedule persistence and 6-9% optimality gap. When $m_1 = 50$, r can take any value from 0 to 30% to guarantee 4.8-6% optimality gap and 78-80% schedule persistence.

5 Conclusion

While there has been recent research on scheduling problems arising in the on-demand travel industry, the issue of schedule persistence has often been neglected by researchers. In this paper, we focused on the rescheduling problem of FMCs. We developed a heuristic to deal with both

Experiment $r, (1-r)n_0, m_0, n_1, m_1$	RRF Opt. Gap (%)	RRF Pers. (%)	RRF No. Charters	ORF Opt. Gap (%)	ORF Pers. (%)	ORF No. Charters	Re-optimization Opt. Gap (%)	Re-optimization Pers. (%)	Re-optimization No. Charters
0.00, 50, 40, 50, 50	4.78%	80.00%	11.4	4.57%	80.00%	11.3	0.00%	62.60%	10.2
0.10, 45, 40, 50, 50	5.81%	79.40%	11.6	4.57%	80.00%	11.3	0.00%	62.60%	10.2
0.20, 40, 40, 50, 50	4.80%	78.00%	10.7	4.57%	80.00%	11.3	0.00%	62.60%	10.2
0.30, 35, 40, 50, 50	7.23%	77.60%	11.4	4.58%	80.00%	11.3	0.00%	62.60%	10.2
0.40, 30, 40, 50, 50	4.47%	76.80%	10.3	4.61%	80.00%	11.3	0.00%	62.60%	10.2
0.50, 25, 40, 50, 50	6.78%	75.20%	10.9	5.12%	79.80%	11.3	0.00%	62.60%	10.2
0.60, 20, 40, 50, 50	6.94%	68.00%	10.8	6.07%	77.40%	11.4	0.00%	62.60%	10.2
0.70, 15, 40, 50, 50	6.79%	57.60%	10.5	4.41%	66.40%	11.1	0.00%	62.60%	10.2
0.80, 10, 40, 50, 50	5.65%	45.40%	10.3	0.98%	53.00%	10.7	0.00%	62.60%	10.2
0.00, 50, 80, 50, 100	8.78%	73.63%	33.0	8.29%	70.75%	32.5	0.00%	50.70%	29.5
0.10, 45, 80, 50, 100	6.96%	72.22%	32.8	8.17%	70.63%	32.4	0.00%	50.70%	29.5
0.20, 40, 80, 50, 100	5.53%	68.38%	31.9	8.21%	70.50%	32.3	0.00%	50.70%	29.5
0.30, 35, 80, 50, 100	5.88%	62.75%	32.6	7.85%	68.75%	32.2	0.00%	50.70%	29.5
0.40, 30, 80, 50, 100	6.95%	55.97%	31.8	7.79%	64.38%	32.2	0.00%	50.70%	29.5
0.50, 25, 80, 50, 100	6.65%	49.88%	32.3	7.38%	56.75%	31.8	0.00%	50.70%	29.5
0.60, 20, 80, 50, 100	5.51%	40.38%	32.1	8.56%	47.75%	32.6	0.00%	50.70%	29.5
0.70, 15, 80, 50, 100	5.31%	29.00%	31.8	4.91%	37.50%	31.5	0.00%	50.70%	29.5
0.80, 10, 80, 50, 100	4.17%	19.63%	31.8	1.49%	25.13%	30.9	0.00%	50.70%	29.5
0.00, 50, 120, 50, 150	6.15%	64.58%	69.1	6.01%	58.50%	69.1	0.00%	48.53%	67.4
0.10, 45, 120, 50, 150	5.01%	59.50%	69.5	5.48%	57.42%	68.9	0.00%	48.53%	67.4
0.20, 40, 120, 50, 150	4.11%	53.67%	68.8	5.15%	55.75%	68.4	0.00%	48.53%	67.4
0.30, 35, 120, 50, 150	4.58%	47.75%	70.2	4.94%	51.83%	68.5	0.00%	48.53%	67.4
0.40, 30, 120, 50, 150	4.02%	40.33%	69.9	4.87%	46.75%	69.4	0.00%	48.53%	67.4
0.50, 25, 120, 50, 150	3.69%	34.67%	69.5	4.62%	40.75%	68.9	0.00%	48.53%	67.4
0.60, 20, 120, 50, 150	3.53%	28.58%	69.3	3.81%	33.58%	69.0	0.00%	48.53%	67.4
0.70, 15, 120, 50, 150	3.27%	20.50%	69.9	3.04%	25.25%	69.1	0.00%	48.53%	67.4
0.80, 10, 120, 50, 150	2.79%	14.33%	68.9	0.02%	16.83%	68.2	0.00%	48.53%	67.4

Table 1: Average performance of re-optimization, randomized reserve-fleet and optimized reserve-fleet heuristics

persistence and optimization issues in scheduling. Our reserve-fleet approach creates a schedule that is flexible enough to deal with future supply and demand changes by enforcing idleness of the aircraft in developing a Master Schedule. We discussed different ways of implementing the reserve-fleet idea. Finally, we conducted computational experiments that quantify the cost-effectiveness vs. persistence of re-scheduling efforts. Based on our numerical results, we discussed how the reserve-fleet size can be determined in a systematic way.

Our novel idea of the reserve-fleet is very practical. We have shown its use for a homogeneous fleet in this paper. Our future work will involve extending this idea to a fleet that includes multiple types of aircraft (where customers can be upgraded/downgraded), and comparing the reserve-fleet heuristics to other standard methods that are used to achieve persistence in optimization (e.g., penalizing deviations from an initial solution). Investigating the effect of both supply and demand uncertainties on the reserve-fleet heuristic also warrants further study.

References

[1] H. Aytug, M.A. Lawley, K. McCay, S. Mohan and R. Uzsoy (2005), Executing production schedules in the face of uncertainties: A review and some future directions, *EJOR* **161**(1), 86 – 110.

[2] F. Bian, E.K. Burke, S. Jain, G. Kendall, G.M. Koole, J.D. Landa Silva, J. Mulder, M.C.E. Paelinck, C. Reeves, I. Rusdi and M.O. Suleman (2003), Making Airline Schedules More Robust, *MISTA 2003 Conference Proceedings*, 678 – 695.

[3] G. Brown, R. Dell and R. Wood (1997), Optimization and Persistence, *Interfaces* **27**(5), 15 – 37.

[4] A. Erdmann, A. Nolte, A. Noltemeier and R. Schrader (2002), Modeling and Solving an Airline Schedule Generation Problem *Annals of Operations Research,* **107**(1), 117 – 142.

[5] D. Espinoza, R. Garcia, M. Goycoolea, G.L. Nemhauser, and M.W.P. Savelsbergh (2006), Per-Seat, On-Demand Air Transportation Part I: Problem Description and an Integer Multi-Commodity Flow Model,Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

[6] D. Espinoza, R. Garcia, M. Goycoolea, G.L. Nemhauser, and M.W.P. Savelsbergh (2006). Per-Seat, On-Demand Air Transportation Part II: Parallel Local Search. Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

[7] J.E. Filar, P. Manhem and K. White (2001), How Airlines and Airports Recover from Schedule Perturbations: A Survey, *Annals of Operations Research,* **108**, 315 – 333.

[8] D. Foust (2007). One Jet, 16 Owners, Big Problems. *BusinessWeek*, January 29, 2007.

[9] R. Hicks, R. Madrid, C. Milligan, R. Pruneau, M. Kanaley, Y. Dumas, B. Lacroix, J. Desrosiers and F. Soumis (2005), Bombardier Flexjet Significantly Improves Its Fractional Aircraft Ownership Operations, *Interfaces* **35**(1), 49 – 60.

[10] I. Karaesmen, P. Keskinocak, S. Tayur and W. Yang (2005), Scheduling Time – Shared Aircraft: Models and Methods for Practice, *Proceedings of MISTA 2005: The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications,* 18 – 21 July, 2005, New York, NY, USA.

[11] P. Keskinocak and S. Tayur (1998), Scheduling of Time – Shared Jet Aircraft, *Transportation Science* **32**(3), 277 – 294.

[12] S. Lan, J.P. Clarke and C. Barnhart (2005), Planning for robust airline operations: optimizing aircraft routings and flight departure times to minimize passenger disruptions. to appear in *Transportation Science.*

[13] C. Martin, D. Jones and P. Keskinocak (2003), Optimizing On-Demand Aircraft Schedules for Fractional Aircraft Operators, *Interfaces* **33**(3), 22 – 35.

[14] National Business Aviation Association (2004), *NBAA Business Aviation Fact Book,* Washington DC, available at *http://web.nbaa.org/public/news/stats/factbook/2004/2004factbook.pdf.*

[15] NetJets (2006), *NetJets Fast Facts.* Woodbridge NJ, December 2006, available at *http//www.netjets.com/News%20and%20Info/pdfs/NetJets%20Fast%20Facts%2012 – 06.pdf.*

[16] D. Ronen (2000), Scheduling Charter Aircraft, *Journal of Operational Research Society* **51**, 258 – 262.

[17] Shebalov, S. and D. Klabjan (2004), Robust airline crew scheduling: move – up crews. *Transportation Science,* **40**, 300 – 312.

[18] Smith, B.C. and E.L. Johnson (2006), Robust Airline Fleet Assignment: Imposing Station Purity Using Station Decomposition. *Transportation Science,* **40**(4), 497 – 516.

[19] G.E. Vieira, J.W. Herrmann and E. Lin (2003). Rescheduling manufacturing systems: a framework of strategies, policies, and methods, *Journal of Scheduling,* **6**(1), 39 – 62.

[20] W. Yang, I. Karaesmen, P. Keskinocak and S. Tayur (2006), Aircraft and Crew Scheduling for Fractional Ownership Programs, Working Paper, Long Island University, Long Island, NY.

[21] Y. Yao, O. Ergun, E. Johnson, W. Schultz and J.M. Singleton (2004), Strategic Planning Problems in Fractional Aircraft Ownership Programs, Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

[22] G. Yu, M. Arguello, G. Song, S. N. McCowan and A. White (2003), A new era for crew recovery at Continental Airlines, *Interfaces,* **33**(1), 5 – 23.

Solving Resource-Constrained Project Scheduling Problem with Particle Swarm Optimization

Sylverin Kemmoé Tchomté, Michel Gourgand, Alain Quilliot
LIMOS, Campus Scientifique des Cézeaux, France, {kemmoe, gourgand, quilliot}@isima.fr

This paper presents a Particle Swarm Optimization (PSO) algorithm for solving Resource-Constrained Project Scheduling Problems (RCPSP). The PSO model is a new population based optimization strategy introduced by Kennedy and Eberhart in 1995. The PSO is a cooperative and competitive algorithm who belongs to the class of the evolutionary algorithms. We here specialize the algorithm of PSO to the particular case of the RCPSP. We have redefined the PSO operators (the addition of two velocities, the difference between two positions, the external multiplication between a scalar and a velocity). We propose in this paper, an extension of the PSO system that integrates a new displacement of the particles and we highlight a relation between the coefficients for each dimension between the classical PSO algorithm and the extension. The experiments on instances from the PSPLIB show that the proposed PSO algorithm is able to solve to optimality most problems of the series and close to the best known solutions on the hardest instances. This seems to indicate that this general algorithm is a good candidate for solving scheduling problems.

Keywords: Particle Swarm Optimization, RCPSP, Evolutionary Algorithms.

1 Introduction

In this paper, we present an algorithm called Particle Swarm Optimization (PSO) for solving the Resource-Constrained Project Scheduling Problem (RCPSP). This problem is interesting for two reasons : on the one hand, it is the core problem of many real-life industrial scheduling applications, where resources can handle several tasks at a time. On the other hand, it is an academic problem for which there exists a variety of algorithms and benchmarks, and new algorithms can be tested against a rigorous gauge. Hence, in order to evaluate a scheduling algorithm designed for particular industrial purposes, if the RCPSP can be recognized in a sub-case of the industrial problem definition, it is interesting measure how well the general algorithm solves the particular RCPSP benchmarks. The PSO algorithm is a stochastic population based optimization approach, first published by Kennedy and Eberhart in 1995. They simulated the scenario in which birds search for food and observed their social behavior. They perceived that in order to find food the individual members determined their velocities by two factors, their own best previous experience and the best experience of all other members. This is similar to the human behavior in making decision where people consider their own best past experience and the best experience of how the other people around them have performed. According to the above concept, Kennedy and Eberhart developed the so-called PSO for optimization of continuous nonlinear functions in 1995.

2 Resource-Constrained Project Scheduling Problem

A Resource-Constrained Project Scheduling Problem (RCPSP) is defined by a set or tasks T and a set of resources R. Each task i is performed on some resources k (possibly many) of which it requires a given amount r_{ik}. Each resource k is available in a given quantity R_k. For each task

i, a duration d_i is given as well as a set of precedence constraints ($i << j$). constraining i to be finished before j starts. All data are integer. The problem is solved when one has found a set of integer starting dates S_i such that :

$$for \quad all \quad precedence \quad constraints \quad with \quad i << j : \quad S_i + d_i \leq S_j \tag{1}$$

$$for \quad all \quad resources \quad k, \quad and \quad all \quad time \quad t : \quad \sum_{\{i|S_i \leq t \leq S_i + d_i\}} r_{ik} \leq R_k \tag{2}$$

The purpose is to minimize the latest end time ($S_i + d_i$) over all tasks i (which is called the makespan of the schedule). A RCPSP problem is known to be NP hard [5] and NP-Complete [6]. As State-of-the-art of the RCPSP, a review of the heuristics can be found in [1]. Notation, classification, models, and methods for the RCPSP are given in [2] and [3].

3 Particle Swarm Optimization algorithm

Particle Swarm Optimization (PSO) is a recently proposed algorithm by J. Kennedy and R. Eberhart in 1995, motivated by social behavior of organisms such as bird flocking and fish schooling. PSO algorithm is not only a tool for optimization, but also a tool for representing socio-cognition of human and artificial agents, based on principles of social behavior. Some scientists suggest that knowledge is optimized by social interaction and thinking is not only private but also interpersonal. PSO as an optimization tool, provides a population-based search procedure in which individuals called particles change their position (state) with time. In a PSO system, particles fly in a multidimensional search space. During flight, each particle adjusts its position according to its own experience, and according to the experience of K neighbors, making use of the best position encountered by itself and its neighbors. The basic PSO model consists of a swarm of particles moving in an n-dimensional, real valued search space of possible problem solutions. For the search space, in general, a certain quality measure, the fitness, is defined making it possible for particles to compare different problem solutions. Every particle i at the time t has the following characteristics: $X_{i,t}$ is the position vector; $V_{i,t}$ is the velocity vector; $P_{i,t}$ is the small memory storing its own best position seen so far; $G_{i,t}$ is the global best position obtained through communication with its fellow neighbor particles. This information flow is obtained by defining a neighborhood topology on the swarm.
The intuition behind the PSO model is that by letting information about good solutions spread out through the swarm, the particles will tend to move to good areas in the search space. At each time step t the velocity is updated and the particle is moved to a new position. This new position is simply calculated as the sum of the previous position and the new velocity:

$$X_{i,t+1} = X_{i,t} + V_{i,t+1} \tag{3}$$

The update of the velocity from the previous velocity to the new velocity is determined by:

$$V_{i,t+1} = \underbrace{c_1 * V_{i,t}}_{momentum} + \underbrace{c_2 * r_2 * (P_{i,t} - X_{i,t})}_{local \quad information} + \underbrace{c_3 * r_3 * (G_{i,t} - X_{i,t})}_{global \quad information} \tag{4}$$

where the parameter c_1 introduced by Shi and Eberhart [7] is the inertia weight, it controls the magnitude of the old velocity $V_{i,t}$. The parameters c_2 and c_3 are real numbers chosen uniformly and at random in a given interval, usually $[0; 1]$. These values determine the significance of $P_{i,t}$ and $G_{i,t}$ respectively. The three velocities components are defined as follows: $c_1 * V_{i,t}$ serves as a momentum term to prevent excessive oscillations in search direction; $c_2 * r_2 * (P_{i,t} - X_{i,t})$ refers to as the cognitive

component, it represents the natural tendency of individuals to return to environments where they experienced their best performance; $c_3 * r_3 * (G_{i,t} - X_{i,t})$ refers to as the social component, it represents the tendency of individuals to follow the success of other individuals.
For the neighborhood system, there are many ways to define it. Kennedy,[8] distinguish two classes: *Physical* neighborhood, which takes distances into account. In practice, distances are recomputed at each time step, which is quite costly, but some clustering techniques need this information. *Social* neighborhood, which just takes *relationships* into account. In practice, for each particle, its neighborhood is defined as a list of particles at the very beginning, and does not change. Note that, when the process converges, a social neighborhood becomes a physical one.

4 Method proposal

We propose a resolution method of the RCPSP problem based on PSO algorithm. This method is an extension of PSO algorithm in which particles have a new displacement technique.

4.1 Proposal of a New Displacement of the particles

The intuition behind this new particle displacement is that by letting information about two good solutions $(P_{i,t}, G_{i,t})$ spread out through the swarm, the particles will tend to move around these solutions in the search space. Let us note by $S_{i,t}$ and $T_{i,t}$ the intermediate positions of particle i at the step t from the position $X_{i,t}$ to the position $X_{i,t+1}$. The figure 1 illustrates of part A this new displacement and the classical one on part B (each co-ordinate of V_{it}, $(P_{it} - X_{it})$ and $(G_{it} - X_{it})$ has its own coefficient).

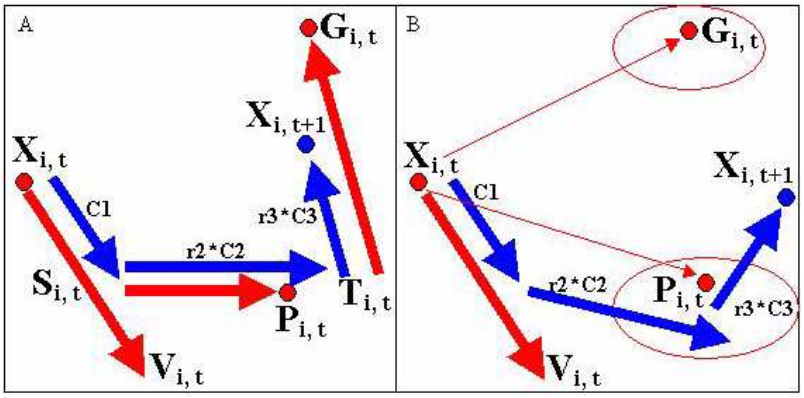


Figure 1: Particle displacement in the swarm space

At each time step t the position is updated, This new position is simply calculated with the intermediate positions, we obtain: $S_{i,t} = X_{i,t} + c_1 * V_{i,t}$; $T_{i,t} = S_{i,t} + c_2 * r_2 * (P_{i,t} - S_{i,t})$; $X_{i,t+1} = T_{i,t} + c_3 * r_3 * (G_{i,t} - T_{i,t}) \implies$

$$V_{i,t+1} = \underbrace{c_1 * (1 - c_2 * r_2) * (1 - c_3 * r_3)}_{\alpha} * V_{i,t} + r_2 * \underbrace{c_2 * (1 - c_3 * r_3)}_{\beta} * (P_{i,t} - X_{i,t}) + r_3 * \underbrace{c_3}_{\gamma} * (G_{i,t} - X_{i,t})$$

$$\text{In short} \quad V_{i,t+1} = \alpha * V_{i,t} + \beta * r_2 * (P_{i,t} - X_{i,t}) + \gamma * r_3 * (G_{i,t} - X_{i,t}) \tag{5}$$

$$\text{with} \begin{cases} \alpha & = c_1 * (1 - c_2 * r_2) * (1 - c_3 * r_3) \\ \beta & = c_2 * (1 - c_3 * r_3) \\ \gamma & = c_3 \end{cases} \quad (6)$$

4.2 Redefinition of PSO operators

The PSO algorithm has been invented to solve continuous non linear functions. Clerc [9] has proposed to extend some operators to solve combinatorial optimization problem (in particular the Traveling Salesman Problem - TSP). For this problem in particular, an exchange of two cities to visit give always a feasible solution but it is not the case of the RCPSP, that contains precedence relations between activities. This is why, while basing us on work of Clerc, we redefine all the operators of the PSO algorithm to treat the case of combinatorial problems with precedence constraints.

- **Position:** A position X is a sequence of activities checking the constraints of precedence ($X = (x_1, ..., x_n)$).
- **Velocity:** A velocity V is a list of couples of activities or transpositions ($V = ((x_i, x_j)_{i,j \in \{1,...,n\}})$). (x_i, x_j) means that exchange activities at the positions x_i^{th} and x_j^{th}. it is possible to have an empty list. The number of transpositions is equal to the norm of the velocity.
- **Opposite of a velocity:** The opposite of a velocity is a velocity. It means to take the same transpositions as in V, but in reverse order. It is easy to verify that the opposite of the opposite velocity give the same velocity.
- **Move of a particle:** the move of a particle is obtained by the application of a velocity to position. The result is another position. It is found by applying the first transposition of V to X, then the second one to the result etc. For example : we consider a position $X = (1, 5, 2, 4, 3)$ with the precedence relation $2 \prec 3$. We consider also a velocity with one transposition (2;5), the activities at the positions 2 and 5 are respectively 5 and 3. Exchange the activities 5 and 3 will violate the relation of precedence between the activities 2 and 3. we will right shift the activity 5 as long as the precedence relation is not violated and we will left shift the activity 3 as long as the precedence relation is not violated. We obtain the position result $X' = (1, 2, 3, 4, 5)$.
- **Difference between two positions:** The difference between two positions is defined as the velocity. To obtain this velocity it is necessary to build an algorithm. We propose a non optimal algorithm that gives this velocity: $NORM$ gives the norm of the velocity obtained, $Exchange()$ is a procedure that allows to permute two activities.
- **Sum of two velocities:** The sum of two velocities V1 and V2 gives a velocity. The list of transpositions which contains first the ones of V1, followed by the ones of V2.
- **External multiplication of a velocity by a scalar:** The external multiplication of a scalar c by velocity V is a velocity V'. This velocity is obtained as follows : If $c = 0$ then $V' = c * V = \emptyset$. If $0 < c < 1$ then the velocity V' is obtained by *truncating* the velocity V and the norm is equal to the integer part of $c * ||V||$. If $c = c' + r > 1$ with c' is an integer number and $0 < r < 1$ a fractional number, we duplicate the velocity V for c' times to obtain V1 and we truncate V according to the number r to obtain V2. the velocity result is the sum of V1 and V2.

Algorithm 1 Difference between two positions X and Y (V = Y - X)

DEBUT
$NORM := 0$
for $j = 1, n$ **do**
  if $X(j) \neq Y(j)$ **then**
    $k := j + 1, trouve := 0$
    while $k \leq n$ AND $trouve = 0$ **do**
      if $X(k) \neq Y(j)$ **then**
        $trouve := 1$
      else
        $k := k + 1$
      end if
    end while
    $NORM := NORM + 1, V \longleftarrow (j, k), Exchange(j, k)$
  end if
end for
END

4.3 Sequence Evaluation

The PSO algorithm allows to update the positions of each particle of the swarm, each position has to be evaluated and a criterion has to be assigned. There is many methods to evaluate a sequence of a combinatorial problem. We propose to use an evaluation algorithm based on non strict order:

Algorithm 2 Sequence evaluation

BEGIN
$ED(job) := 1 \quad \forall job$
for $j = 1, n$ **do**
  $job := X(j), t := ED(job), trouve := 0$
  while $trouve = 0$ **do**
    $start := t, end := t + PT(job) - 1$
    if $ResourceAvailable(job, start, end)$ **then**
      $trouve := 1$
    else
      $t := t + 1$
    end if
  end while
  $ST(job) := deb - 1, ET(job) := ST(job) + PT(job), ResCons(job)$
  for $\forall jobsuc \in Succ(job)$ **do**
    $ED(jobsuc) := ET(job)$
  end for
end for
END

$ED(job)$ is a vector that gives for an activity called job the earliest date of its beginning; $PT(job)$ is a vector that gives for an activity called job the processing time; $ST(job)$ is a vector that gives for an activity called job the start time; $ET(job)$ is a vector that gives for an activity called job the end time; $ResCons(job)$ is a procedure that allows the resource consumption by the

activity called job; $Succ(job)$ is a procedure that gives all the immediate successors of the job.

5 Computational Analysis

This section reports on a computational comparison of several of the heuristics. For test instances, we have used the standard sets j30 and j60 provided on the website (http://129.187.106.231/psplib). The projects in these test sets consist of 30 and 60 activities, respectively and consist of 480 instances each (10 replications). The number of generated and evaluated schedules are fixed to 1000 and 5000, respectively. Tables 1 and 2 present a comparison of PSO to state-ofthe-art heuristic algorithms. We have looked at all the algorithms referred to in [11]. The coefficients ($\alpha = 0.047, \beta = 0.378, \gamma = 1.494$) used in this paper are obtained by applying equation (6) on those ($c_1 = 0.729, c_2 = 1.494, c_3 = 1.494$) provided by [21]. According to [9], the best swarm and neighbourhood sizes are depending on the number of local minimums of the objective function. Usually, we simply do not have this information, except the fact that there are at most N-1 such minimums. In this study, we fix the swarm size at N particles where N the number of dimension of the search space. Table 1 summarises the average deviations from optimal makespan in j30. The results indicate that PSO is capable of providing near optimal solutions. The average CPU time for PSO is 0.2 and 2.18 seconds for 1000 and 5000 evaluations respectively, what are very competitive.

Algorithm type	Authors	Iterations 1000	Iterations 5000
PSO	**Kemmoe et al.**	**0.26**	**0.21**
GA, TS, path reli.,[10]	Kochetov and Stolyar (2003)	0.1	0.04
Hybrid GA, [11]	Valls et al. (2007)	0.27	0.06
GA, [12]	Alcaraz and Maroto (2001)	0.33	0.12
DJGA, [13]	Valls et al. (2005)	0.34	0.20
Sampling + BF/FB, [14]	Tormos and Lova (2003a)	0.25	0.13
Tabu Search, [16]	Nonobe and Ibaraki (2002)	0.46	0.16
Sampling + BF, [17]	Tormos and Lova (2001)	0.30	0.16
Self-adapting GA, [19]	Hartmann (2002)	0.38	0.22
Activity list GA, [18]	Hartmann (1998)	0.54	0.25
Sampling + BF, [15]	Tormos and Lova (2003b)	0.3	0.17

Table 1: Average deviations from optimal solution - J=30

Table 2 summarises the average deviations from critical path based lower bound in j60. The critical path makespan lower bound is obtained by computing the length of a critical path in the resource relaxation of the problem. The results indicate that our algorithm outperforms all algorithms in block one for all schedule limits. PSO is capable of providing near optimal solutions. The average CPU time for PSO is 1.77 and 6.18 seconds for 1000 and 5000 evaluations respectively.

Algorithm type	Authors	Iterations 1000	Iterations 5000
PSO	**Kemmoe et al.**	**9.52**	**9.01**
Hybrid GA	Valls et al.	11.56	11.10
GA, TS, path reli.,[10]	Kochetov and Stolyar (2003)	11.71 11.17	
DJGA, [13]	Valls et al. (2005)	12.21	11.27
Self-adapting GA, [19]	Hartmann (2002)	12.21	11.70

Activity list GA, [18]	Hartmann (1998)	12.68	11.89
Sampling + BF/FB, [14]	Tormos and Lova (2003a)	11.88	11.62
Sampling + BF, [15]	Tormos and Lova (2003b)	12.14	11.82
GA, [12]	Alcaraz and Maroto (2001)	12.57	11.86
Sampling + BF, [17]	Tormos and Lova (2001)	12.18	11.87
SA, [20]	Bouleimen and Lecocq (2003)	12.75	11.90

Table 2: Average deviations from critical path - J=60

6 Conclusion

We have proposed in this paper, a new particle displacement for PSO. We suggest that the proposal extension of PSO balance better the process of exploitation (intensification) and exploration (diversification). We have developed a mathematical formulation which preserves the methodical mind of PSO. This formulation provides a correspondence between the coefficients of our proposal method and those of the classical PSO.
We have applied this extension of PSO to the combinatorial optimization problem of RCPSP. These problems contain constraints of precedence and the general framework provided by clerc [9] can not be applied. We propose in this paper to adapt this framework to hold prcedence constraints by redefining the operators to obtain only feasible sequences.
The computational results show that PSO is a fast and high quality algorithm. Our algorithm is competitive with other state-of-the-art heuristics for the instance set j30. For the instance set j60, our algorithm outperforms all state-of-the-art algorithms for the RCPSP.
The results quality can still be increased if we add the two mechanisms regularly present in the published metaheuristics : an initial population (swarm) built by an heuristic and a local search method exploiting the specific knowledge of the problem (critical path method for example).
The future research of this paper is to provide more computational analysis of this extension on j90 and j120 sets of RCPSP and on others combinatorial optimization problems (Flow-Shop, Job-Shop, etc).

References

[1] S. Hartmann, and R. Kolisch (1998), Experimental Evaluation of State-of-the-art Heuristics of Resource-Constrained Scheduling Problem, *Invited paper of the INFORMS spring conference - Cincinatti may 2-5*, N 476.

[2] P. Brucker, and A. Drexl, and R. Mohring, and K. Neumann, E. Pesch (1999), Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* **112**, 3 – 41.

[3] R. Kolisch, and R. Padman (2001), An integrated survey of deterministic project scheduling, *Omega* **29**(3), 249 – 272.

[4] P. Brucker (1999), An integrated survey of deterministic project scheduling, *Omega* **29**(3), 249 – 272.

[5] J. Blazewicz, J.K Lentra, et A.H.G Rinnoy Kan (1983), Scheduling projects subject to resource constraintes: classification and complexity, *Discrete Applied Mathematics* **5**, 11 – 24.

[6] P.Brucker, S. Knust, D. Roper, and Y. Zinder (1999), Scheduling UET task system with concurrency on two parallel identical processors, *Mathematical Methods of Operation Research*.

[7] Y. Shi and R. Eberhart (1998), Parameter Selection in Particle Swarm Optimization. *Annual conference Evolutionary programing*, San Diego.

[8] J. Kennedy (1999), Small worlds and megaminds: Effects of neighborhood topology on particle swarm performance, *Proceedings of the IEEE Congress on Evolutionary Computation*, 1931 – 1938.

[9] M. Clerc (2004), Discrete Particle Swarm Optimization, In G.C. Onwubolu and B.V. Babu, editors, *New Optimization Techniques in Engineering*, Springer-Verlag, 219 – 204.

[10] A. Kochetov and Y. Stolyar (2003), Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem, In: *Proceeding of Workshop on Computer Science and Information Technologies*.

[11] V. Valls and F. Ballestin and S. Quintanilla (2007), A hybrid genetic algorithm for the resource-constrained scheduling problem, *European Journal of Operational Research*, doi:10.1016/j.ejor.2006.12.033

[12] J. Alcaraz and C. Maroto (2001), A robust genetic algorithm for resource allocation in project scheduling, *Annals of Operations Research* **102**, 83 - 109.

[13] V. Valls and F. Ballestin and S. Quintanilla (2005), *Justification and RCPSP: A technique that pays*, *European Journal of Operational Research* **165**, 375 - 386.

[14] P. Tormos and A. Lova (2003a), Integrating heuristics for resource constrained project scheduling: One step forward, Technical report, Department of Statistics and Operations Research, Universidad Politekcnica de Valencia.

[15] P. Tormos and A. Lova (2003b), An efficient multi-pass heuristic for project scheduling with constrained resources, *International Journal of Production Research* **41**, 1071 - 1086.

[16] K. Nonobe and T. Ibaraki (2002), Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP). In: Ribeiro, C.C., Hansen, P. (Eds.), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, 557 - 588.

[17] P. Tormos and A. Lova (2001), A competitive heuristic solution technique for resource-constrained project scheduling, Annals of Operations Research **102**, 65 - 81.

[18] S. Hartmann (1998), A competitive genetic algorithm for resource-constrained project scheduling, Naval Research Logistics **45**, 733 - 750.

[19] S. Hartmann (2002), A self-adapting genetic algorithm for project scheduling under resource constraints, *Naval Research Logistics* **49**, 433 - 448.

[20] K. Bouleimen and H. Lecocq (2003), A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, *European Journal of Operational Research* **149**(2), 268 - 281.

[21] M. Clerc (1999), The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. Congress on Evolutionary Computation, Washington, DC, Piscataway, NJ: IEEE Service Center., 1951 - 1957.

Analyzing Basic Representation Choices in Oversubscribed Scheduling Problems

Laurence A. Kramer, Laura V. Barbulescu and Stephen F. Smith
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, USA
{lkramer, laurabar, sfs}@cs.cmu.edu

Both direct schedule representations as well as indirect permutation-based representations in conjunction with schedule builders are successfully employed in oversubscribed scheduling research. Recent work has indicated that in some domains, searching the space of permutations as opposed to the schedule space itself can be more productive in maximizing the number of scheduled tasks. On the other hand, research in domains where task priority is treated as a hard constraint has shown the effectiveness of local repair methods that operate directly on the schedule representation. In this paper, we investigate the comparative leverage provided by techniques that exploit these alternative representations (and search spaces) in this latter oversubscribed scheduling context. We find that an inherent difficulty in specifying a permutation-based search procedure is making the tradeoff in guaranteeing that priority is enforced while giving the search sufficient flexibility to progress. Nonetheless, with some effort spent in tuning the move operator, we show that a permutation-space technique can perform quite well on this class of problem in cases of low oversubscription and in fact was able to find new optimal solutions to a few previously published benchmark problems. Not surprisingly, the permutation-space search does not perform as well as the schedule-space search in terms of maintaining schedule stability.

Keywords: Heuristic Search, Local Search, Meta-heuristic Search.

1 Introduction

In The Sciences of the Artificial [14], Herbert Simon recounts the example of an ant making its way toward its eventual goal over a wave-molded beach. The ant's path seems somewhat random and somewhat directed, with a complexity that is not easily decipherable. Simon makes the point that the ant itself does not navigate by some complex algorithm, but that the apparent complexity arises from the features of the terrain it traverses. In a like vein, as we craft search techniques to navigate a complex space, it has often proven fruitful to employ knowledge about the contours of the space in the search process. It is generally accepted in AI that choosing the right representation for a problem can result in search spaces in which the solutions are easier to find.

Our specific focus in this paper is on oversubscribed scheduling problems – scheduling problems for which there is typically more to do than available resources will allow and the search problem is to determine which subset of activities to include. For this class of problem, scheduling research has successfully exploited the use of both direct schedule representations and indirect, usually permutation-based, representations that can be expanded into direct representations as a basis for solution. However, very little research has directly analyzed the implications and tradeoffs of this representational choice for solving various oversubscribed scheduling problems. Barbulescu et al.[1] have shown a permutation-based genetic algorithm to outperform a schedule repair based technique in the context of scheduling groundstation access for satellite communications. Globus et al.[6, 5] similarly argue the relative superiority of a permutation-based representation over a direct representation based on Gantt charts for scheduling requests for time on Earth observing satellites.

In this paper, we consider the basic question of representation choice in the context of another oversubscribed scheduling domain, the US Air Force Air Mobility Command (AMC) scheduling problem [11]. The AMC problem is distinguishable from most other oversubscribed scheduling problems studied in the literature in one major respect: task priority is treated as a hard constraint. This characteristic models a range of real-life situations, especially in military operations, where the priority of a task trumps all other considerations and is not subject to tradeoff. It can be contrasted with the more typical treatment of priority in oversubscribed scheduling, where the search problem is formulated as one of optimizing a weighted sum of the priorities of all scheduled tasks and it is possible to trade one higher priority task for some number of lower priority tasks. Repair-based algorithms working in the schedule space can be designed to enforce this hard constraint. In fact, the *TaskSwap* procedure originally developed for solving the AMC problem is such a schedule-space search algorithm [9, 10]. The question we consider in this paper is whether the relative superiority of permutation-based techniques in other oversubscribed domains carries over to domains like the AMC problem where task priority is a hard constraint.

We hypothesized that it would be difficult to design a competitive search algorithm for the AMC domain using a permutation-based representation, mainly because of the inherent difficulties of enforcing the task priority constraint. The hypothesis was based on what we thought to be a reasonable idea: searching for solutions directly in the schedule space, as opposed to using an indirect representation which needs to be translated into a schedule, should be better able to adhere to the hard task priority constraint, while at the same time focusing the search to produce good solutions faster. We found that, after a certain amount of tuning, a permutation-based search technique *can* perform comparably with the repair-based technique for low to moderate levels of oversubscription. However, as the problems become more oversubscribed, the permutation-based technique is outperformed by the schedule-space technique. For problems with high levels of oversubscription there are very few opportunities to move things around, and the permutation-based technique fails to identify them.

2 Basic Representational Choices

For many scheduling applications in general and oversubscribed scheduling applications in particular, the representation of choice is a permutation of the tasks to be scheduled, together with a schedule builder to transform the permutation into a schedule [16, 15, 6, 1]. The advantage of using such an indirect representation is that a wide range of general-purpose search algorithms (from heuristic methods to exact, tree search methods) can be employed to search the permutation space, while all the particular constraints of the domain are encapsulated in a schedule builder. As a downside, in general, one can not predict the effect of a change in the current solution (permutation) until the schedule builder computes the new schedule. Also, depending on the schedule builder, the set of schedules that can be reached from the permutation space is usually a proper subset of all possible schedules. It is not clear if this subset contains the optimal solution. Finally, the permutation search space/schedule builder combination is not well suited for preserving schedule stability (since it is difficult to predict how a permutation change will affect the corresponding schedule).

Searching the schedule space directly can be an attractive alternative ([7, 3, 9, 10]) to permutation space search. Powerful domain-specific heuristics are usually available (for example, various resource contention measures), and such measures can drive the search. Also, when schedule stability is important, search operators for the schedule space can be defined such that stability is preserved (by minimally changing the current schedule). While general-purpose search operators can still be defined, efficient search algorithms in the schedule space will typically exploit domain

knowledge to decide how to reorganize the schedule. The challenge is in defining the right search operators.

3 The AMC Scheduling Domain

The AMC scheduling problem considered in this paper abstracts the large airlift and tanker mission scheduling problem faced by the US Air Force Air Mobility Command (USAF AMC). Drawing from [11], the problem is briefly summarized as follows:

- A set T of tasks (or missions) are submitted for execution. Each task $i \in T$ has an earliest pickup time est_i, a latest delivery time lft_t, a pickup location $orig_i$, a dropoff location $dest_i$, a duration d_i and a priority pr_i
- A set Res of resources (or air wings) are available for assignment to missions. Each resource $r \in Res$ has capacity $cap_r \geq 1$ (corresponding to the number of aircraft for that wing).
- Each task i has an associated set Res_i of feasible resources (or air wings), any of which can be assigned to carry out i. Any given task i requires 1 unit of capacity (i.e., 1 aircraft) of the resource r that is assigned to perform it.
- Each resource r has a designated location $home_r$. For a given task i, each resource $r \in Res_i$ requires a positioning time $pos_{r,i}$ to travel from $home_r$ to $orig_i$, and a de-positioning time $depos_{r,i}$ to travel from $dest_i$ back to $home_r$.

A schedule is a *feasible* assignment of missions to wings. To be feasible, each task i must be scheduled to execute within its $[est_i, lft_i]$ interval, and for each resource r and time point t, *assigned-cap*$_{r,t} \leq cap_r$. Typically, the problem is over-subscribed and only a subset of tasks in T can be feasibly accommodated. If all tasks cannot be scheduled, preference is given to higher priority tasks.

3.1 AMC Task Priority as a Hard Constraint

In the AMC domain tasks are scheduled strictly by priority. Furthermore, it is not normal procedure to "trade off" or bump a higher priority mission to get some number of lower priority missions into the schedule. There are five major priority classes: 1 through 5, with 1 the highest priority class.

A simple way of producing schedules that satisfy the task priority constraint is to assign prospective tasks in priority order. Unless the schedule is extremely oversubscribed, tasks in the highest priority classes will tend to be feasibly assigned. Assuming that the problem is oversubscribed, though, at some point resource unavailability will not allow the next pending task to be inserted. Subsequent (lower priority) tasks can still be feasibly assigned if resources are available for the time periods required. Hence it is possible for a schedule satisfying the priority constraint to include tasks of lower priority than those of some tasks that have been forced to be excluded.

A solution (schedule) S respects the hard priority constraint if the following two rules hold:

1. There exists no feasible solution S' that contains a superset of the priority 1 tasks that are contained in S. In other words, there is no S' where all priority 1 tasks that are assigned to resources in S *and* one or more additional priority 1 tasks (unassigned in S) are assigned.[1]
2. For any $k = 1..4$, there exists no S' that contains exactly the same set of assigned tasks at priority levels 1 through k that are contained in S and a superset of the priority $k+1$ tasks that are contained in S.

[1]This does not imply that S' should schedule exactly the same tasks at the other lower priority levels; in fact it is likely that the scheduled lower priority tasks would be different.

An optimal solution then is a solution satisfying the 2 rules above that also minimizes the number of unassigned tasks at each level. We define S to be *hard-priority optimal* if there is no solution with exactly the same number of unassignables in the first k highest priority classes and fewer unassignables in the $k+1$th (next lower) priority class, for any $k = 0..4$.

3.2 Enforcing the Hard Priority Constraint

In practice, given that the problem we consider is $\mathcal{NP}$-complete [4], it is not in general feasibly possible to prove that a solution is hard-priority optimal, let alone that the above two rules for enforcing the task priority are respected. Instead, we choose to define a new objective function that amplifies the differences between priority classes. Given a schedule, any change that would insert into the schedule a lower priority unassignable by unscheduling (bumping) a higher priority task is precluded by the new objective function.

More specifically, we resort to using a heuristic *scoring value* for each priority class, that emphasizes the differences between classes: priority 5 maps to 1, priority 4 to 1,000, priority 3 to 1,000,000, and so on. We define the *penalty score* for a schedule as the sum of the scoring values for all the unassignables. The new objective function minimizes the penalty score. Since the number of tasks we consider is less than 1,000, by using a factor of 1,000 between successive priority classes, we ensure that a higher priority task couldn't possibly be swapped out to make way for any number of lower priority tasks and still achieve a better (lower) penalty score.

4 Algorithms

Previous research has found *TaskSwap*, a repair-based algorithm, to perform well on the AMC scheduling problem [9, 10, 12]. Conceptually, the *TaskSwap* procedure proceeds by temporarily relaxing the priority constraint, retracting one or more scheduled tasks (regardless of priority) to allow insertion of a previously unassignable task, and then recursively attempting to reintroduce retracted tasks elsewhere into the schedule. We define a variant of this base procedure in Section 4.2 as our representative schedule-space search procedure for the experimental analysis to follow.

A wide range of algorithms searching in permutation space have been implemented for oversubscribed scheduling domains. One stands out because of its similarity with *TaskSwap*: Squeaky Wheel Optimization (SWO) [8]. *SWO* temporarily assigns higher priority to unscheduled tasks and attempts to schedule them earlier. Also, *SWO* has been shown to be one of the best performing algorithms for another oversubscribed domain, scheduling groundstation access for satellite communications [2]. Hence we choose *SWO* as our representative permutation-space search procedure, defined in Section 4.3. We also implemented various hill-climbing algorithms, employing several simple swap operators as well as a "temperature descent swap" [6] operator, but found these techniques to perform rather poorly when running for a reasonable number of iterations.

4.1 The Schedule Builder

The schedule builder produces an initial schedule for both *TaskSwap* and *SWO*; also, it translates from the permutation search space to the schedule space while running *SWO*. Given a permutation of tasks, the schedule builder considers the tasks in the order in which they appear in the permutation and uses a look-ahead heuristic to assign start times and resources to tasks, based on predicted resource contention. This heuristic, *max-availability*, is described in some detail in [12]. To produce the initial schedule, the schedule builder is applied to the tasks sorted in priority order.

4.2 TaskSwap

As indicated above, the *TaskSwap* scheduling procedure of [9, 10] implements a repair-based approach to rearranging tasks in an input schedule so as to include additional, previously unassignable tasks. The algorithm considers unassignable tasks one by one (in priority order) and attempts to insert them in the existing schedule by retracting some number of conflicting tasks. The retracted tasks are reassigned subsequent to assigning the formerly unassignable task, the idea being that there might exist a new feasible schedule where retracted tasks are shifted somewhat in time or assigned to an alternate resource. The algorithm recurses on any retracted task that cannot be reassigned, and returns successfully when all visited tasks are assigned, or with failure when all tasks contending for the same set of alternate resources have been considered. For a complete algorithmic description of this basic *TaskSwap* procedure, we direct the reader to [9].

TaskSwap localizes its search for a solution that enforces the hard priority constraints by inserting new unassignable tasks into the schedule without unscheduling any higher priority tasks in the process. In fact, *TaskSwap* can be seen to take an overly conservative approach to enforcing the priority constraint; since it will never unschedule a task that is in its input schedule, it can actually miss opportunities for an improved solution that might result from substituting one task of a given priority for another task of the same priority.

Noticing this deficiency, and also to take advantage of the more general objective function approximation of the priority constraint defined earlier, we define a less constraining version of *TaskSwap*, where an unassignable task u is considered successfully inserted and the new solution is accepted even when some initially scheduled tasks of lower priority than u are not reinserted back into the schedule, if the sum of the scoring values for these tasks is lower than the scoring value of u. We call this version *PriorityTaskSwap*. It is easy to see that the following property holds:

Property 1 *The final schedule produced by the new version of* PriorityTaskSwap *can never result in a worse (higher) penalty score than the initial schedule.*

4.3 Squeaky Wheel Optimization (SWO)

SWO [8] repeatedly iterates through a cycle composed of three phases. First, a scheduling order of the elements in the problem is defined and based on this a greedy solution is built. Then, the solution is analyzed and the elements causing "trouble" are ranked based on their contribution to the objective function. Third, the ranking of such "trouble makers" is used to move them earlier in the scheduling order. The cycle is repeated until a termination condition is met.

In our scheduling context, we build the initial greedy solution based on the priorities associated with the elements in the problem. The "trouble maker" elements are unassigned tasks. During each iteration, we examine the schedule and identify the unassignable tasks. Based on its priority, we associate a move distance with each unassignable task, which indicates how far forward in the scheduling order this task will move for the next iteration.

5 Experimental Design

For our experiments with *PriorityTaskSwap* and *SWO*, we used the 100-instance data set of mission scheduling problems for the AMC domain which is described in [11]. Each problem instance consists of approximately 1000 missions (tasks) that must be scheduled across 14 air wings (resources) with variable capacity. The 100 instances are divided into 5 sets of 20, each being progressively more resource constrained. For each instance an initial schedule is generated after which *PriorityTaskSwap* and *SWO* are run. For *SWO* we found empirically that moving unassignable tasks

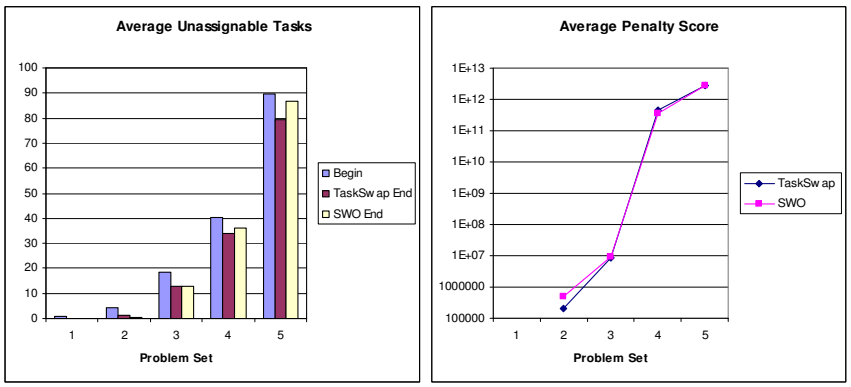


Figure 1: Solution Quality: Number of Unassignable Tasks and Penalty Score

forward for only a short distance (50 to 100 positions) did not perform well. Setting the distance to around 200 resulted in best performance. We defined the distance to move forward an unassignable u as: $200 + (10 \times numeric\_priority(u))$, where $numeric\_priority(u) = 6 - priority\_class(u)$.[2] For each instance, *PriorityTaskSwap* is run to completion, and *SWO* is run until an iteration limit of 50 is reached[3] or the procedure finds an optimal solution (0 unassignable tasks).

6 Experimental Results

First, we compare *SWO* and *TaskSwap* in terms of the penalty score and end number of unassignables (Figure 1). We find that the two algorithms perform similarly for the first four sets of problems. A Wilcoxon signed-rank test [13] shows that the results are not significantly different for these problems. *SWO* is able to find two new optimal solutions (zero unassignable tasks), for problems 4 and 20 in set 3 [11]. However, for the problems with high levels of oversubscription (in set 5), a Wilcoxon signed-rank test shows that *TaskSwap* finds significantly better solutions in terms of number of unassignables. *TaskSwap* is able to make progress on the harder problems by selectively focusing on a relatively small subset of tasks, for which it is productive to temporarily relax the priority constraint, and concentrating on moving these tasks. In contrast, *SWO* is unable to find the appropriate combination of tasks to move in order to achieve comparable results.

While *TaskSwap* focuses on which tasks to move around in order to improve the solution, *SWO* is unconstrained in terms of what it can move around in the schedule. We conjecture that this flexibility of *SWO* is the reason it finds new optimal solutions for problems with low levels of oversubscription and at the same time, this is what hurts its performance for moderate to high levels of oversubscription. This conjecture is supported by our schedule stability results.

To assess schedule stability, we compare initial and final schedules for each run in terms of: number of tasks that shifted in time and number of tasks that changed resource (Figure 2). As expected, *SWO* performs poorly in terms of schedule stability. When the level of oversubscription is low, there is also more freedom in rearranging the schedule to fit in more of the initial unassignables. *SWO* takes advantage of this: on average it changes the start time for as many as 700 tasks for

[2]For example, a priority-1 unassignable task will be move forward 250 positions, and a priority-2 task 240 positions.

[3]SWO typically found its best solution after a few iterations, and running more than 50 iterations produced no further improvement. We also experimented with stochastic versions of the priority to distance mapping, but were unable to achieve better results.

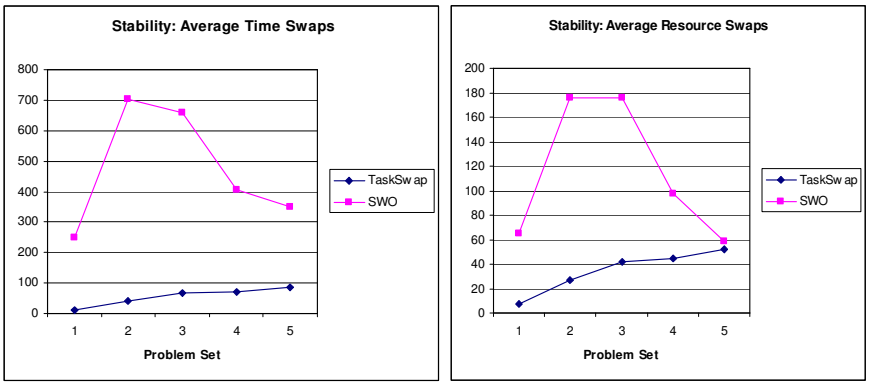


Figure 2: Solution Stability: Number of Time and Resource Swaps

the problem set 2 and reassigns resources to more than 170 tasks for that set. For high levels of oversubscription, there isn't much flexibility in rearranging tasks in the schedule (since the initial schedule is so packed). *SWO* moves a lot fewer tasks for problem sets 4 and 5 than it does for 2 and 3, and it cannot find the right set of moves to perform as well as *TaskSwap*.

7 Final Remarks

Implementing a permutation-based algorithm that respects task priority constraints is a challenging proposition. Our approach in this paper has been to define an objective function that encodes the priority constraint as a numeric sum, that can only be improved by assigning more tasks of higher priority. The difficulty though is the inherent problem of guaranteeing that priority is enforced while giving the search sufficient flexibility to progress.

Overall, we find two main results with respect to choice of representation for solving oversubscribed scheduling problems that require task priority to be enforced as a hard constraint. First, for low levels of oversubscription, a permutation-based search technique is able to take advantage of the many possibilities that exist to rearrange currently scheduled tasks to incorporate new unassigned tasks into the schedule; for such problems, a permutation-based representation can work as well (or even outperform) a repair-based technique. Second, for moderate to high levels of oversubscription, there is little room to move tasks around in the schedule; a technique like *TaskSwap* that focuses exactly on which tasks would need to be reassigned has a much better chance of finding improvements than a permutation-based technique, which lacks such guidance.

Acknowledgements

This research was supported in part by the USAF Air Mobility Command under Contract # 7500007485 to Northrop Grumman Corporation, by the Department of Defense Advance Research Projects Agency (DARPA) under Contract # FA8750-05-C-0033, and by the CMU Robotics Institute. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the USAF or DARPA.

References

[1] L. Barbulescu, A. Howe, and L. D. Whitley (2006), AFSCN scheduling: How the problem and solution have evolved. *M*CM **43**, 1023 – 1037.

[2] L. Barbulescu, L. D. Whitley, and A.E. Howe (2004), Leap before you look: An effective strategy in an oversubscribed problem, *P*roc. 19th National Artificial Intelligence Conference.

[3] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran (2000), ASPEN - Automating space mission operations using automated planning and scheduling, *6*th International SpaceOps Symposium (Space Operations), Toulouse (France).

[4] M. S. Garey and D. S. Johnson (1979), *C*omputers And Intractability: A Guide To The Theory Of NP-Completeness, W.H. Freeman and Company, New York.

[5] A. Globus, J. Crawford, J. Lohn, and R. Morris (2002), Scheduling earth observing fleets using evolutionary algorithms: Problem description and approach, *P*roc. of the 3rd Int'l NASA Workshop on Planning and Scheduling for Space, 27 – 29.

[6] A. Globus, J. Crawford, J. Lohn, and A. Pryor (2004), A comparison of techniques for scheduling earth observing satellites, *P*roc. of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), Sixteenth Conference on Innovative Applications of Artificial Intelligence (IAAI-04), 836 – 843.

[7] M. D. Johnston and G. Miller (1994), Spike: Intelligent scheduling of hubble space telescope observations, *I*ntelligent Scheduling, M. Zweben and M. Fox editors, Morgan Kaufmann.

[8] D. E. Joslin and D. P. Clements (1999), "Squeaky Wheel" Optimization, *J*AIR **10**, 353 – 373.

[9] L. A. Kramer and S. F. Smith (2003), Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems, *P*roc. 18th Int'l Joint Conf. on AI, Acapulco Mexico.

[10] L. A. Kramer and S. F. Smith (2004), Task swapping for schedule improvement, a broader analysis, *P*roc. 14th Int'l Conf. on Automated Planning and Scheduling, Whistler BC.

[11] L. A. Kramer and S. F. Smith (2005), The amc scheduling problem: A description for reproducibility, Technical Report CMU-RI-TR-05-75, *Robotics Institute, Carnegie Mellon University.*

[12] L. A. Kramer and S. F. Smith (2005), Maximizing availability: A commitment heuristic for oversubscribed scheduling problems, *P*roc. 15th International Conference on Automated Planning and Scheduling (ICAPS-05), Monterey CA.

[13] R. Ott and M. Longnecker (2000) *An Introduction to Statistical Methods and Data Analysis, 5th Ed.*, Duxbury Pr.

[14] H. A. Simon (1981), *T*he Sciences of the Artificial, 2nd Edition, The MIT Press.

[15] G. Syswerda (1991), Schedule Optimization Using Genetic Algorithms, Lawrence Davis, editor, *H*andbook of Genetic Algorithms, chapter 21. Van Nostrand Reinhold, NY.

[16] L. D. Whitley, T. Starkweather, and D. Fuquay (1989), Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator, J. D. Schaffer, editor, *P*roc. of the 3rd Int'l. Conf. on GAs, Morgan Kaufmann.

A Case Study of an Integer Programming Model for Instructor Assignments and Scheduling Problem

Eddie Cheng, Serge Kruk[1]
Department of Mathematics and Statistics, Oakland University, Rochester, MI 48309,USA,
{echeng, kruk}@oakland.edu

We express an integer program formulation of the rostering problem academic departments face every semester. The goal is to model the real situation, with all the relevant exceptions and peculiarities and to optimize the instructor satisfaction level with their timetable. We introduce *Preference Sets* and *Preference Pairs* as the main abstraction on which the model is based. Abstraction that can be used to allow the subjective satisfaction level of instructors to be modeled satisfactorily. We describe an instance in some detail and comment on the surprisingly short solving time required by modern integer program solvers for a type of problem that, until recently, was considered too hard for this type of formulation. The Department of Mathematics and Statistics at Oakland University has adopted this formulation since 2006.

Keywords: Rostering, Case-study, Model.

1 Introduction

Timetabling course sections, students, rooms and instructors in higher education is an area that attracted much research. Every two years the Conference on the Practice and Theory of Automated Timetabling (`www.asap.cs.nott.ac.uk/patat/patat-index.shtml`) [6, 3, 5, 4, 8, 7] will see a number of papers on the subject. [2, 14] give recent research directions in this area and [13] argues very convincingly, by combing the publications of the last decades, that a large gap exists between the theory and the actual implementations used in institutions. Some recent case studies include [12, 10, 9, 11]. In such a timetabling problem, the emphasis is on scheduling courses to satisfy room constraints, sometimes some student constraints are included, but in almost all cases instructors' concern is treated only very partly if it is treated at all. For example, the authors of [9, 11] have an approach similar to ours, but only attempts at satisfying a request for a specific time period or course while satisfying other timetabling constraints. We will model many more types of requests since we are not concerned with room and time assignments. The authors of [1] model the faculty satisfaction problem in a manner similar to ours but their criteria are somewhat different and some of them are simpler.

At some universities, the individual unit, such as the Department of Mathematics and Statistics at Oakland University, has little or no control over the room assignment and has limited control over scheduling of section time. For example, one cannot change the section time of Calculus without considering section time of non-math sections such as engineering sections as most of the students in Calculus are engineering majors. Hence any timetabling project aimed at some form of global optimization will be interdepartmental. This, usually, means such project will not get started and timetabling is done piecemeal, often by hand.

In this setting, at Oakland University, the limited responsibility of Department of Mathematics and Statistics (DMS) is reduced to assigning courses to its faculty members. This task is traditionally performed by the associate chair manually. Due to a recent administrative realignment of

resources, the position of associate chair was eliminated. We approached the chair of the department and suggested that we cast the problem as an optimization problem where the objective is to maximize the "faculty happiness." So this optimization task is to find teaching schedule for instructors. Given that the problem no longer involves timetabling of courses, a considerable amount of effort can be spent on maximizing such "faculty happiness." This is now the primary focus rather than a secondary focus. We can now include many desires of the instructors into the model.

Somewhat more precisely, the problem is: given a number of courses with fixed credit values (typically 1 to 5), each with possibly multiple sections (1 to 10) meeting at set times of the week and given a number of instructors with fixed teaching loads in measured in number of credits (typically 0 to 9) and with subjective preferences for courses, for days of the week, for times of day, assign instructors to courses to maximize the satisfaction of the subjective preferences.

As in a typical operations research project, the first phase is to test our formulation; this is done by trial runs in assigning courses for Fall 2005 and Winter 2006. We then gather suggestions from faculty members to refine our formulation. It turns out that for most faculty members, their main concern is the teaching time as oppose to the actual courses. In particular, almost all prefer not to have a 5-day schedule so that they can devote at least one full day for research. Moreover, they like the teaching schedule to complement their service schedule. Finally, we implemented the new formulation in assigning courses for Fall 2006 and Winter 2007. This paper is not theoretical in nature, and no theoretical result is offered. This is a case study in using operations research to solve a real problem that has received little attention in the past. Moreover, the model we offer can be included into a more general timetabling model to allow some consideration of the satisfaction of instructors in addition to the usual time and space constraints.

2 Problem to solve

Every semester at Oakland University, each department is faced with the problem of assigning instructors to sections of courses, which we will, henceforth call *sections*. These sections have already assigned time-slots. Rooms will be later assigned by the registrar's office to account for enrollment. The available timeslots are divided into two main categories. Monday-Wednesday-Friday schedule and Tuesday-Thursday schedule. The first meet only on Monday, Wednesday and Friday and the second on Tuesday and Thursday. Table 1 exemplifies some of the course offerings in Fall 2006.

MTH154	Section 1	MWF 1040 to 1147
MTH154	Section 2	MWF 1200 to 1307
MTH154	Section 3	MWF 1320 to 1427
MTH154	Section 4	TR 1730 to 1917
MTH155	Section 1	MWF 0800 to 0907
MTH155	Section 2	MWF 1040 to 1147
MTH155	Section 3	TR 1730 to 1917

Table 1: Course Offering

In this example, MTH 154 Calculus I has four sections and MTH 155 Calculus II has three sections. This set of all sections can be partitioned into subsets where each subset corresponds to a course. So the subset MTH 154 is of size 4 and the subset MTH 155 is of size 3. The complete list of timeslots is given in Table 2.

1	MWF	0800	0907	2	MWF	0920	1027	3	MWF	1040	1147
4	MWF	1200	1307	5	MWF	1320	1427	6	MWF	1440	1547
7	MW	1530	1717	8	TR	0800	0947	9	TR	1000	1147
10	TR	1300	1447	11	TR	1530	1717	12	MW	1730	1917
13	TR	1730	1917	14	MW	1930	2117	15	TR	1930	2117

Table 2: Timeslots

In our department, we have about twenty-eight full-time faculty members. However, due to sabbatical leaves, usually about twenty-five are active in any given semester. A number of sections are covered by adjunct faculty members and graduate students. Each regular member typically teaches two courses per semester. The traditional procedure is for each faculty member to send the associate chair a list of his/her favourite courses and a list of courses he/she want to stay away from. The faculty member also indicates his/her time-of-day and day-of-week preferences. For example "only afternoons" or "only Monday-Wednesday-Friday" sections. Historically, a number of variations on these preferences were expressed. From these the associate chair constructs a schedule by assigning instructors to sections by hand. Often such an assignment is obtained from modifying the teaching schedule from the year before.

Our job was to automate this process and improve upon it. Part of the work involved developing the database manager and access code to enter the preferences. We will not discuss this aspect of the project here but will rather concentrate on the operations research aspect of the work, the development of an integer linear programming model to automate the assignment. To improve upon this process, the objective function is to maximize faculty *satisfaction* (a fuzzy concept to be defined later). Specifically, we address the following issues that we deem deficient in the traditional process.

1. There is neither ordinal nor cardinal relationship between a course preference and a time-slot preference.

2. The relationship in the list of favourite courses for an instructor is only ordinal and not cardinal.

3. Although modifying the teaching schedule from the year before produces a good schedule, some instructors are pigeonholed into a couple of courses.

3 Integer linear model

3.1 Preferences and input data

The input is a list of sections (a section, is an instance of a course, meeting at specific times of the week, with a value of a certain number of credits) and a list of instructors (each with a list of preferences) and a charge of a certain number of credits to teach.

Each instructor indicates his preferences via weights attached to sections in various direct and indirect ways. In the most direct manner, preferences provided by each instructor include a weight for each course. Note that there may be more than one section of a given course. The instructor, in this first list, is indicating his preference for subject matter, not time of the day or day of the week.

We model this first list of preference as a weight $wc_i^c \in \mathbb{R}$, for each section c, for instructor i. We allocate the weight given to a course to each of its sections. Note that preferences can be positive,

indicating an attraction for the given section or negative, indicating a repulsion. The absence of a preference for section c will imply $wc_i^c = 0$, a "don't care" indication.

To model time preferences of instructor, we introduce the concept of *preference sets*, S_j for $j \in \{0, \ldots, N_{PS}\}$ in the model, where N_{PS} is the number of preference sets. Those are subsets of all sections related by time. Typical instances of these preference sets are enumerated in the Table 3. For example, all the sections that have a meeting at 8 in the morning are in S_0. So `MTH 154` Section 1 is in S_1 and S_4 only. Instructor i indicates a preference for each S_j and the weight $ws_i^j \in \mathbb{R}$ is distributed to every section in S_j.

j	S_j
0	0800h sections
1	Morning sections (0800h – 1159h)
2	Afternoon sections (1200h – 1729h)
3	Night sections (1730h – 2100h)
4	Monday-Wednesday-Friday sections
5	Tuesday-Thursday sections
6	Friday sections
7	1930h sections

Table 3: Preference sets.

To model more complex requests, we introduce *preference pairs*, P_j for $j \in \{0, \ldots, N_{PP}\}$ sets of pairs of sections, which we illustrate in the Table 4. (Again, N_{PP} denotes the number of preference pairs.) For example, the pair (`MTH 154` Section 4, `MTH 155` Section 1) is in S_1 and the pair (`MTH 155` Section 1, `MTH 155` Section 3) is in S_0. Instructor i indicates a preference for each P_i and its weight ($wp_i^j \in \mathbb{R}$) is distributed to every element of P_j.

j	P_j
0	Morning and night sections on same day
1	Night section of one day followed by morning section of the next day
2	Monday and Tuesday sections
3	Consecutive sections
4	Three consecutive timeslots
5	A class then free timeslot, then a class

Table 4: Preference pairs.

Notice that the three categories of user weights, the course preference weights, the set preference weights and the pairs preference weights must satisfy the following for each instructor $i \in I$,

$$\sum_{c \in C} |wc_i^c| + \sum_{j=0}^{N_{PS}} |ws_i^j| + \sum_{j=0}^{N_{PP}} |wp_i^j| = 1.$$

This has a normalizing effect on the weights of all instructors. (The users are not required to satisfy this constraint manually when entering their preferences, of course. We simply scale the weights as we generate the integer program.)

3.2 Decision variables

The main index sets are C, I and T. The complete list of sections, C is partitioned into k subsets $C_1, \ldots C_k$ each representing the sections of a given course. The set of instructors is I and the set of possible timeslots is T.

The decision variable is

$$x_{ic} = \begin{cases} 1, & \text{if instructor } i \in I \text{ is in section } c \in C \\ 0, & \text{otherwise.} \end{cases}$$

To simplify some expressions, we introduce two indicator variables. This is done at no computational cost since they are simple sums of the decision variables. From (1a),we see that z_c, when non-zero, indicates whether section c is assigned to any instructor. From (1b), we see that y_{it} indicates whether an instructor is busy at a given time.

$$z_c := \sum_{i \in I} x_{ic} \qquad \forall c \in C \tag{1a}$$

$$y_{it} := \sum_{c \in X_t} x_{ic} \qquad \forall i \in I, \forall t \in T \tag{1b}$$

where X_t is the set of all the sections in timeslot t.

3.3 Hard constraints

3.3.1 One instructor per section

Each section needs only one instructor.

$$\sum_{i} x_{ic} \leq 1 \qquad \forall c \in C. \tag{2}$$

3.3.2 Course leaders

Some multi-section courses are handled by adjunct (which we do not schedule) with the exception that one faculty member must be leader and teach one of the sections. This is handled by extracting subsets of the sections C_j for $j \in I^L$, the index set of the partitions requiring leaders. The model constraint is then

$$\sum_{c \in C_j} z_c \geq 1 \qquad \forall j \in I^L. \tag{3}$$

3.3.3 Load constraint

Each instructor has to teach a certain total number of credits indicated by the lower bound L_i and the upper bound U_i. Each course has a certain number of credits indicated by M_c. The model constraints to ensure that the exact number of credit is met for each instructor is

$$L_i \leq \sum_{c} M_c x_{ic} \leq U_i \qquad \forall i \in I \tag{4}$$

3.3.4 No cloning constraint

Each instructor can lecture only one section during a given timeslot.

$$y_{it} \leq 1 \qquad \forall i \in I, \forall t \in T. \tag{5}$$

But unfortunately, our schedule includes some overlapping timeslots: a 1440h-1547h as well as a 1530h-1717h. To take this into consideration, we construct a set of pairs of overlapping timeslots O and model the constraint as

$$x_{ic_1} + x_{ic_2} \leq 1 \qquad \forall i \in I, \forall (c_1, c_2) \in O. \tag{6}$$

3.3.5 Incompetence constraint

Preventing certain instructors from being assigned certain courses is of course, trivial. Assuming instructor i is deemed incompetent to teach section c, we generate the constraint

$$x_{ic} = 0. \tag{7}$$

3.4 Soft Constraints

Since the major goal of the model is to maximize instructor satisfaction, understood as giving them, as far as possible, both the courses they want and the times they want, the following constraints are part of the objective function.

3.4.1 Preference set

First, we introduce a non-negative variable r_i^j defined by

$$r_i^j = \sum_{s \in S_j} y_{is} \qquad \forall i \in I, j \in \{0, \ldots, N_{PS}\}. \tag{8}$$

Note that r_i^j counts the number of instances where instructor i is assigned a section in preference set S_j. Assuming that the instructor allocates a weight ws_i^j to a preference set S_j, the following term $ws_i^j \times r_i^j$ in a maximizing objective function will try to increase r_i^j if the weight is positive and reduce it to 0 if the weight is negative.

3.4.2 Preference pair

The preference pairs are handled similarly. First, we introduce non-negative variables rr_{kl}^j and rrr_i^j. The former, as seen in (9a)-(9c), indicates whether sections k and l of preference pair P_j are assigned to instructor i.

$$y_{ik} + y_{il} - rr_{ikl}^j \leq 1 \qquad \forall i \in I, \forall (k,l) \in P_j, \forall j \in \{0, \ldots, N_{PP}\}, \tag{9a}$$

$$rr_{ikl}^j \leq y_{ik} \qquad \forall i \in I, \forall (k,l) \in P_j, \forall j \in \{0, \ldots, N_{PP}\}, \tag{9b}$$

$$rr_{ikl}^j \leq y_{il} \qquad \forall i \in I, \forall (k,l) \in P_j, \forall j \in \{0, \ldots, N_{PP}\}, \tag{9c}$$

$$rrr_i^j = \sum_{(k,l) \in P_j} rr_{ikl}^j \qquad \forall i \in I, \forall j \in \{0, \ldots, N_{PP}\}. \tag{9d}$$

The latter, as seen in (9d), counts the number of such assignments correspond to preference pair P_j. If the instructor allocates a weight wp_i^j to a set of preference pairs P_j, we then add the following term $wp_i^j \times rrr_i^j$ into the objective function

3.5 Objective function

Since covering our upper-level courses is a department priority, we have a weight wf_c of non-negative value for each course — with service courses having value 0. This allows some flexibility in the assignation of courses, independently of the preferences of the faculty members. This also potentially leaves lower-level courses open for to adjunct which we schedule later, when we know exactly which sections need to be covered.

The goal is to maximize satisfaction of faculty preferences and department preferences, modeled as

$$\max \sum_{i \in I} \sum_{c \in C} x_{ic} wc_i^c + \sum_{i \in I} \sum_{j=0}^{N_{PS}} ws_i^j r_i^j + \sum_{i \in I} \sum_{j=0}^{N_{PP}} wp_i^j rrr_i^j - \sum_{c \in C} wf_c(1 - z_c) \tag{10}$$

4 Experimental results

The model was written in `GNU Mathprog`[2] and has been used on a real schedule for the first time during the Fall semester 2005 and every semester since then. The complete model, as well as some typical datasets are available at `http://homepage.oakland.edu/~kruk/research` In a typical semester, the number of instructors is almost 30, the number of sections is a little below 100. Recall that we do not allocate an instructor to all sections as the left-over positions will be manned by part-timers and graduate students. The resulting number of variables in a typical instance is around 5000 with about half being integer and the number of constraints hovers just under 10000. Even on an old Pentium computer, most instances are solved in seconds or minutes with all the solvers tried (CPLEX[3], Cbc[4] and glpk[5]).

There are notable variations in difficulty from one semester to the next. In CPLEX, the presolver will eliminate between 10% and 30% of the rows and columns. The branch and bound code will never solve at the root node but will visit from 10 to 1000 nodes with the gap being reduced from 5% down to zero; so that even the first integer solution is likely to be acceptable in practice if the solution time to prove optimality ever became a problem. Most of the effective cuts are generic Gomory fractional cuts, on the order of a few dozens, with some cover cuts in addition, for some instances.

5 Conclusion

By the subjective reaction of the chair of the department and our colleagues, the implementation of this system has been a success. This is in contrast to the usual litany of complaints following a hand-crafted schedule. This is not entirely surprising since preferences, even if they were known with certainty, were handled in a ad-hoc fashion, hence often forgotten. More to the point, with an interface allowing each instructor to put weights on each combination of courses and times, the true relative importance of each choice was clearly spelled out and taken into consideration. Some instructors in a certain semester want a specific course with no concern for times; during other semesters, the time preferences trump any choice of courses. The model allows complete flexibility in this manner. Even more telling is that, after the initial run, we were asked to add a few more preferences (related to two sections during three consecutive timeslots), which the model accommodates trivially, suggesting that it is fairly robust to change. The program is now adopted by the department as a permanent procedure.

That all solvers solve the problem very rapidly is somewhat surprising considering the number of integer variables and constraints. This is partly due to the advances in general-purpose solvers, undoubtedly, but not exclusively. All the solvers produce different solutions every semester. This multiplicity of optimal solutions plays to our advantage and indicates that a two-phase approach to the university timetabling problem, with the first phase solely concerned with students and

[2] `http://www.gnu.org/software/glpk/glpk.html`
[3] http://www.ilog.com/products/cplex/
[4] http://www.coin-or.org/Cbc/index.html
[5] http://www.gnu.org/software/glpk/

room requirements while the second caters to instructor's preference is a natural and efficient decomposition of the problem. This may be the more interesting result of the experiment since it suggest that adding a varied and flexible list of instructor preferences to a timetabling model will likely not unduly affect the solving time.

6 Acknowledgements

The authors would like to thank the anonymous referees for valuable comments and for pointing out some pertinent case studies.

References

[1] S.M. Al-Yakoob and H.D. Sherali (2006) Mathematical programming models and algorithms for a class-faculty assignment problem. *E*uropean J. Oper. Res. **2**(173), 488 – 507.

[2] E. Burke and S. Petrovic (2002) Recent research directions in automated timetabling. *Eur. J. Oper. Res.* **140**(2), 266 – 280.

[3] E.K. Burke and M.W. Carter, editors (1998) The Practice and Theory of Automated Timetabling II: Selected papers from the 2nd International conference, *Lecture Notes in Computer Science* **1408**, Springer.

[4] E.K. Burke and P. De Causmaecker, editors (2003) The Practice and Theory of Automated Timetabling IV: Selected papers from the 4th International conference, *Lecture Notes in Computer Science* **2740**, Springer.

[5] E.K. Burke and W. Erben, editors (2000) The Practice and Theory of Automated Timetabling III: Selected papers from the 3rd International conference, *Lecture Notes in Computer Science*, **2079**, Springer.

[6] E.K. Burke and P. Ross, editors (1996) The Practice and Theory of Automated Timetabling: Selected papers from the 1st International conference, *Lecture Notes in Computer Science* **1153**, Springer.

[7] E.K. Burke and H. Rudová, editors (2006) The Practice and Theory of Automated Timetabling VI: Selected papers from the 6th International conference, ISBN 80-210-3726-1, Brno, Czech Republic, Masaryk University.

[8] E.K. Burke and M. Trick, editors (2005) The Practice and Theory of Automated Timetabling V: Selected papers from the 5th International conference, *Lecture Notes in Computer Science* **3616**, Springer.

[9] S. Daskalaki, T. Birbas, and E. Housos (2004) An integer programming formulation for a case study in university timetabling, *European J. Oper. Res.* **153**(1), 117 – 135.

[10] T. Hinkin and G. Thompson (2002) Schedulexpert: Scheduling courses in the cornell university school of hotel administration, *Interfaces*, **32**(6), 45 – 57.

[11] Badri M. A.and Davis D. L., Davis D. F., and Hollingsworth J (1998) A multi-objective course scheduling model : Combining faculty preferences for courses and times, *Computers and operations research*, **25**(4), 303 – 316.

[12] C. Martin (2004) Ohio university's college of business uses integer programming to schedule classes, *Interfaces*, **34**(6), 460 – 465.

[13] Barry McCollum (2006), University timetabling: Bridging the gap between research and practive, *Burke and Rudová* [7], 15 – 35.

[14] A. Schaerf and L. Di Gaspero (2006), Timetabling research: State-of-the-art and discussion, *Burke and Rudová* [7], 52 – 62.

Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems

Philippe Laborie, Daniel Godard
ILOG, 9 rue de Verdun, 94253 Gentilly, France, {plaborie, dgodard}@ilog.fr

Providing robust scheduling algorithms that can solve a large variety of scheduling problems with good performance is one of the biggest challenge of practical schedulers today. In this paper we present a robust scheduling algorithm based on Self-Adapting Large Neighborhood Search and apply it to a large panel of single-mode scheduling problems. The approach combines Large Neighborhood Search with a portfolio of neighborhoods and completion strategies together with Machine Learning techniques to converge on the most efficient neighborhoods and completion strategies for the problem being solved. The algorithm is evaluated on a set of 21 scheduling benchmarks, most of which are well established in the scheduling community. Despite the generality of the approach, for 17 benchmarks out of 21, its mean relative distance to state-of-the-art problem specific algorithms is less than 4%. It even outperforms state-of-the-art problem-specific algorithms on 7 benchmarks clearly showing that our algorithm offers a valuable compromise between robustness and performance.

Keywords: Constraint Logic Programming, Heuristic Search, Local Search, Machine Scheduling, Meta-heuristic Search, Production Scheduling, Real World Scheduling, Shop-Floor Scheduling.

1 Introduction

There exists a large variety of scheduling problems and scheduling applications each of them featuring different types of resources, different types of temporal network topology (jobs, precedence network, work breakdown structure), different objective functions, etc. Facing this variability, the scheduling literature is huge. Most of it is about identifying or providing theoretical or experimental results on a particular type of scheduling problem. For a given well identified problem, for instance the job-shop scheduling or resource-constrained project scheduling (RCPSP) problems, extremely efficient optimization algorithms are available. Experimental evaluations are usually based on a set of specific benchmarks for the problem being studied which also explains the large number of benchmarks available to the scheduling community.

Still, when one is faced with a practical scheduling application the gap between the problem to be solved and state-of-the-art problem specific algorithms is usually too large. It requires an advanced expertise in scheduling to assess their potential applicability and efficiency on the real problem and to adapt them when possible. It explains why actual scheduling applications tend to use in-house heuristics rather than very efficient but often too specific optimization algorithms.

Providing robust scheduling algorithms that can solve a large variety of scheduling problems with good performance is still a challenge. In this paper we present a robust scheduling algorithm based on Self-Adapting Large Neighborhood Search. Section 2 describes the class of scheduling problem we focus on which covers a large panel of single-mode scheduling problems. The algorithm itself is presented in section 3. Section 4 reports an experimental study over 21 scheduling benchmarks most of which are well established in the scheduling community (e.g. job-shop, RCPSP). We show among other things that for 17 benchmarks out of 21, the mean relative distance to state-of-

the-art problem-specific algorithms is less than 4% and that our approach, despite its generality, outperforms the state-of-the-art on 7 benchmarks.

2 Model

Although the algorithm described in section 3 can easily be extended to handle complex scheduling problems involving, for instance, multiple modes, resource minimal capacities or calendars, we focus in this paper on its application to a more restricted but still expressive class of single-mode scheduling problems involving the following features:

- *Non-preemptive activities of fixed or variable duration.* $\mathcal{A} = \{A_1, ..., A_n\}$ denotes the set of activities of the schedule. Each activity can be specified a release date (minimal start time) and a deadline (maximal end time).
- *General temporal network.* If x_i and x_j denote two time-points (start or end time of some activity), any temporal constraint $x_i - x_j \in [D_{ij}^{min}, D_{ij}^{max}]$, D_{ij}^{min}, $D_{ij}^{max} \in \mathbb{Z}$ can be expressed.
- *Capacity resources (unary, discrete, discrete reservoir) with maximal profiles.* Discrete resources are renewable resources of limited capacity, discrete reservoirs are resources of limited capacity that can be produced and consumed by activities [24]. Each capacity resource R_k can be associated a function $C_k : \mathbb{Z} \rightarrow \mathbb{Z}^+$ that represents its maximal capacity over time.
- *State resources.* State resources are resources whose state can change over time. Two activities requiring a given state resource to be in a different state cannot overlap in time [26].
- *Sequence-dependent setup times on unary resources.* A setup time on a unary resource R_k is specified by a setup matrix M_k with $M_k[i, j] \in \mathbb{Z}^+$ denoting the minimal time that must elapse between the end of A_i and the start of A_j when A_j is executed next to A_i on R_k.
- *Cost expressed as a sum/max aggregation of semi-convex piecewise linear (SCPL) functions on start, end and duration of activities.* A semi-convex function [21] is a function such that, if one draws a horizontal line anywhere in the Cartesian plane corresponding to the graph of the function, the set of x such that $f(x)$ is below the line forms a single interval. Some examples of SCPL functions are depicted on Figure 1.

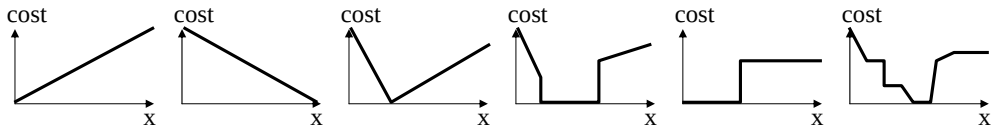


Figure 1: Example of semi-convex piecewise linear functions

Most of the classical single-mode scheduling problems (e.g. job-shop, RCPSP) and classical cost functions (e.g. makespan, earliness/tardiness costs, weighted number of late jobs, duration minimization or maximization, etc) can be represented using this model.

Note also that in this paper we focus on optimization problems rather than on feasibility problems. We assume that finding an initial feasible solution (even of bad quality) is not hard.

3 Self-Adapting Large Neighborhood Search

3.1 Overview

Large Neighborhood Search (LNS) [34] is based upon a process of continual relaxation and re-optimization: a first solution is computed and iteratively improved. Each iteration consists of a relaxation step followed by a re-optimization of the relaxed solution. This process continues until some condition is satisfied, typically, when a time limit is reached. In this paper, we generalize the randomized LNS proposed in [17] along two directions: (1) the scope of the approach is dramatically enlarged, now handling a wide variety of resource types and cost functions, and (2) the approach is robustified by using portfolios of large neighborhoods and completion strategies in combination with Machine Learning techniques to converge on the most efficient neighborhoods and completion strategies for the problem being solved.

The overall framework of Self-Adapting LNS (denoted SA-LNS) is illustrated on Figure 2. Each large neighborhood LN_i and each completion strategy CS_j in the portfolios are associated a vector of parameters. In a parameter vector $p = (p^1, ..., p^n)$, each parameter p^k takes its values in a finite set V^k. The learning algorithm maintains two probability distributions $prob(LN_i)$ and $prob(CS_j)$ on the sets of large neighborhoods and completion strategies and, for each parameter p^k, a probability distribution on its possible values in V^k. At each cycle of the LNS, one large neighborhood LN_i together with a corresponding vector of parameter values P_i and one completion strategy CS_j with a corresponding vector of parameter values Q_j are selected based on the current probability distributions. $LN_i[P_i]$ is applied to relax the current best solution then, completion strategy $CS_j[Q_j]$ is applied to re-optimize the relaxed solution. After this cycle, LN_i and CS_j, together with their respective parameter values P_i and Q_j are rewarded according to the efficiency of the cycle defined by the ratio $r = \Delta c/\Delta t$ where Δc is the cost improvement if any (0 otherwise) and Δt is the cycle CPU time. This type of reward tends to favor neighborhoods and strategies that quickly converge on good solutions. The reward increases the probability of the rewarded elements being chosen according to a classical re-enforcement scheme: $weight_{t+1} = (1 - \alpha) \cdot weight_t + \alpha \cdot r$ with learning rate $\alpha \in (0, 1]$ being a parameter of the global approach.

In [12], the authors present an algorithm switching strategy that iteratively runs the whole set of algorithms in the portfolio and adapts, at each cycle, their allocated running times depending on their past results. SA-LNS learns the algorithm selection rather than the algorithm running times which allows for a more fine-grain control of the search and avoids systematically running useless algorithms. The overall framework is actually closer to the one recently described in [31] for Vehicle Routing problems, the main difference being that the our approach also learns the parameter values of each component of the LNS (neighborhoods, completion strategies). That way, it can be seen as a pure black-box search without any parameter.

The sequel of this section describes the portfolios of large neighborhoods and completion strategies. Note that the first solution is built using the same search strategy as the completion strategy *SetJustInTime* described in section 3.3.

3.2 Large neighborhoods

The portfolio of large neighborhoods currently consists of 3 neighborhoods. They are all based on the initial generation of a Partial Order Schedule (POS) [32] constructed from a completely instantiated solution where activities have fixed start times and end times. A POS is a directed graph G($\mathcal{A}$, $\mathcal{E}$) where the edges in $\mathcal{E}$ are precedence constraints between activities with the property that any temporal solution to the graph is also a resource-feasible solution. Algorithms for transforming

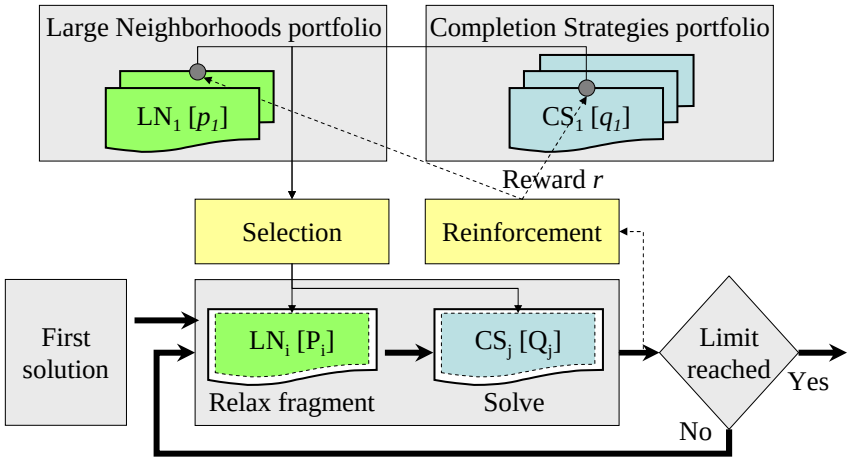


Figure 2: Self-Adapting LNS overview

a fully instantiated solution into a POS are described in [32, 17]. We extend this approach to state resources and discrete reservoirs as sketched below.

- *State resource.* The POS $P(R_k)$ of a state resource R_k contains all the edges $A_i \rightarrow A_j$ such that activities A_i and A_j require incompatible states of R_k and A_i is executed before A_j in the solution.
- *Discrete reservoirs.* The algorithm to generate a POS $P(R_k)$ for discrete reservoirs works in two steps: one that ensures that the reservoir does not underflow and the other that it does not overflow. In the first step, a simple pegging heuristic chronologically creates a directed graph of pegging arcs between producing activities and consuming activities: the first producer is pegged to the first consumer and the pegged quantity is the minimum between the produced quantity and the consumed quantity. The process continues until all consuming activities are provided enough quantity. Let $P_u(R_k)$ be the graph of pegging arcs. In the second step, the pegging arcs are used to build a sub-model to ensure that the reservoir does not overflow: each pegging arc is represented by an activity that requires the pegged quantity of a discrete resource R'_k whose capacity is the maximum capacity of the reservoir. The algorithm described in [17] is applied on this discrete resource to build a POS $P_o(R'_k)$. The POS of the discrete reservoir $P(R_k)$ is then defined as: $P(R_k) = P_u(R_k) \cup P_o(R'_k)$.

The global POS P is defined as $P = \cup_k P(R_k)$. Redundant edges in P are removed. The goal of the neighborhoods is to *select* a subset of activities that will be relaxed in the POS P. As described in [17], the relaxed POS P' is obtained by removing from P all the edges involving at least one selected activity and adding new edges to repair broken paths. The relaxed POS P' is then used to enforce precedence constraints between activities before applying a completion strategy. The portfolio contains the 3 following neighborhoods:

- *RandomizedNHood*$[\alpha_R]$. This is the neighborhood described in [17]. It randomly selects activities with a probability α_R, where α_R is a self-adapting parameter of the neighborhood.
- *TimeWindowNHood*$[\alpha_W, \beta_W]$. Activities are first sorted by increasing start times in an array. The selected activities are those whose index in the sorted array belongs to $[\beta_W \cdot n, (\beta_W + \alpha_W) \cdot n]$ where n is the number of activities of the problem, and α_W and β_W are two self-adapting parameters.

- *TopologicalNHood*$[\alpha_T, \beta_T]$. This neighborhood is similar to the previous one. It only differs in the ordering of activities. The activities are sorted in the following lexicographic order: increasing connected component[1] (CC) indexes, increasing strongly connected component (SCC) indexes, increasing start times. α_T and β_T are two self-adapting parameters. This neighborhood tends to select activities belonging to the same CC (resp. SCC) of the problem.

3.3 Completion strategies

Currently, only one completion strategy *SetJustInTime*$[\gamma]$ is used. This completion strategy explores a search tree with a maximal number of failures equal to $\gamma \cdot n$ where n is the number of activities of the problem and γ is a self-adapting parameter. At the root node, this strategy solves a linear relaxation of the problem that only takes into account activity durations, temporal constraints and a convexification of the SCPL functions of the cost. The optimal solution of this relaxation gives indicative start and end times for each activity. The search is a generalization of the *SetTimes* strategy recapped in [17]. It considers activities by increasing indicative start times and tries to schedule them as close as possible to their indicative times. When a failure occurs, in the right branch, the activity is marked "unselectable" and will remain so until constraint propagation removes from the current domain of the activity the start or end dates that were tried on the left branch. When the objective is regular, this strategy boils down to *SetTimes*. At each LNS step, the completion tries to find a solution that is not worse than the current best solution in term of cost value. If the maximal number of failures is reached before such a solution is found, a new move is tried.

4 Experimental study

SA-LNS has been implemented on top of ILOG CP 1.1 using ILOG CPLEX 10.1 for the linear relaxation of the *SetJustInTime* strategy. We report in this section a comparison of this implementation with state-of-the-art specialized algorithms on 21 scheduling benchmarks, most of which are well established in the scheduling community. It is to be noted that for this experimental study, we consider our method as a pure black-box: there is no tuning of the search for the different benchmarks. The results are summarized[2] on Table 1. When possible, we compare with the upper-bounds (UB) found by the best specialized algorithm on each benchmark (column "Reference UB") and try to use comparable time limits, otherwise we compare with the best known upper-bounds (which may have been found by different algorithms) and use a time-limit which is a piecewise linear function of the number n of activities, for instance $1800s$ on a 2GHz laptop for a problem with 500 activities. Note that due to the number of benchmarks, we often had to select a subset of instances. To ensure a fair comparison, these instances were randomly drawn. Column "MRD" measures the average relative distance to the reference upper-bound, a negative value means that in average SA-LNS outperforms the reference algorithm(s). The number of selected instances, together with the number of improved upper bounds compared to the reference algorithm(s) is given in the last column.

Over the 21 benchmarks, the worse average distance of SA-LNS is 14.7% on single-machine problems with common due-date which can be considered as reasonable for a generic approach that do not exploit problem specificities. In fact, the two worse results are single machine problems

[1] Connected components and strongly connected components of the temporal network are computed from the initial set of temporal constraints in the problem. They convey important information about the temporal structuration of the problem (jobs, work breakdown structure, etc.).

[2] The detailed experimental protocol and results are available on `scheduler.ilog.fr`.

Problem type	Benchmark	Problem size	Reference UB	MRD	# Imp. UBs / # Instances
Trolley	[41]	230-460	[19]	−11.8%	15/15
Hybrid flow-shop	[35]	200-1000	[35]	−8.8%	19/20
Job-shop w/ E/T	[3]	30-200	[3]	−5.6%	32/48
Air traffic management	[19]	2000	[19]	−4.0%	1/1
Flow-shop w/ E/T	[27]	30-400	[14]	−3.0%	4/12
Max. quality RCPSP	[33]	30	[33]	−2.3%	NA/3600
Cumulative job-shop	[28]	150-675	[17]	−0.3%	27/86
Single proc. tardiness	[20]	200-500	[20]	0.2%	0/20
Semiconductor testing	[30]	400	[30]	0.2%	7/18
RCPSP w/ E/T	[42]	30-50	[42]	0.4%	15/60
Open-shop	[9, 40, 18]	64-400	[15, 7, 25]	0.9%	0/28
RCPSP	[23]	120	Best PSPLIB	1.6%	0/600[3]
Shop w/ setup times	[10]	50-200	[2]	2.3%	0/15
Parallel machine w/ E/T	[29]	8-200	[4]	2.6%	2/52
Job-shop	[1, 39, 43, 40]	100-500	Best OR-Lib	2.8%	0/33
Air land	[5]	10-50	[5]	3.4%	0/8
Flow-shop w/ buffers	[40]	100-500	[8]	3.6%	12/30
Flow-shop	[40]	100-500	Best OR-Lib	5.9%	0/22
Aircraft assembly	[16]	575	[13]	8.7%	0/1
Single machine w/ E/T	[11, 37, 29]	8-500	[38]	9.8%	1/100
Common due-date	[6]	100-200	[36]	14.7%	0/20

Table 1: Results of SA-LNS on 21 scheduling benchmarks

without any precedence constraint and SA-LNS currently does not perform any special treatment for unary resources (except for the constraint propagation done in CP1.1). Except for those two very specific scheduling benchmarks, SA-LNS is always less than 9% away from the best performing approaches and for 17 benchmarks out of 21, the mean relative distance is even less than 4%. This illustrates the exceptional robustness of the approach. Moreover SA-LNS outperforms the state-of-the-art on 7 benchmarks which is remarkable given the generality of the approach. For 10 benchmarks (all the ones for which the number of improved upper-bounds with respect to the reference is positive except *Flow-shop w/ E/T* for which we improve on the reference but not on the best bounds of the literature) SA-LNS is able to improve some best known upper bounds ever reported.

The present approach is a generalization of the randomized LNS described and experimented in [17] on cumulative job-shop problems. An interesting fact is that, in spite of this generalization, SA-LNS outperforms the results reported in [17]: 27 instances out of 86 have been improved, 9 have been worsen and the MRD to randomized LNS is −0.3%. On this benchmark, 12 of the best known upper bounds have been improved. It illustrates the added value of online learning which allows automatically tuning the algorithm on the actual instance being solved.

[3]The average deviation from the path-based lower bound is 32.4%. We estimate the average number of LNS cycles to be slightly less than 50000 which would position SA-LNS in the top 7 best approaches for RCPSP among the 37 approaches reviewed in [22].

5 Conclusion and future work

The Self-Adapting Large Neighborhood Search presented in this paper combines several ingredients which are fundamental to its efficiency and robustness:

- *Large Neighborhood Search*: by freezing some features of a solution and focusing on re-optimizing the unfrozen features the LNS framework provides a general and efficient traversal of the search space. Compared with Tree Search, it avoids being stuck with wrong early decisions. It is more flexible than Local Search for complex problems involving many types of constraints and resources.
- *Partial Order Schedules*: in the context of LNS, POSs provide a very powerful way to inject flexibility into the schedule while keeping interesting features from one solution to the other. As shown, the concept can be extended to various types of resources.
- *Neighborhoods*: taken individually, each of the neighborhoods described in the paper are fairly robust (see for instance [17] for *RandomizedNHood*).
- *Completion strategy*: the *SetJustInTime* completion strategy uses a linear relaxation of the problem and, doing so, has a global vision of the ideal position of activities in time would there be no resource limitation. In the context of LNS where only a part of the POS is unfrozen, this relaxation tends to be very informative as most of the resource constraints are still captured by frozen precedence arcs of the POS. The branching scheme of the strategy allows to exploit constraint propagation and better explore the bottom of the search tree which clearly is a plus compared to more classical non-backtracking greedy algorithms.
- *Learning*: the re-enforcement learning scheme, although quite simple, ensures a quick convergence on the most effective neighborhoods, completion strategies and their associated parameter values. Learning is a key factor in the robustness of the approach.

On-going and future work mainly consist in extending SA-LNS to multi-mode scheduling problems, that is, scheduling problems that have a "resource allocation" dimension (this also accounts for optional activities, alternative resources, alternative recipes or routes, etc.) and to other types of costs such as setup, resource usage or inventory costs.

References

[1] J. Adams, E. Balas, and D. Zawack (1988), The shifting bottleneck procedure for job shop scheduling, *Management Science* **34**(3), 391 – 401.

[2] E. Balas, N. Simonetti, and A. Vazacopoulos (2005), Job shop scheduling with setup-times, deadlines and precedence constraints, In *Proc. MISTA-2005*.

[3] P. Baptiste, M. Flamini, and F. Sourd (2005), Lagrangean bounds and lagrangean heuristics for just in time job-shop scheduling, Technical report, Universita degli Studi di Roma Tre.

[4] P. Baptiste and R. Sadykov (2007), A new mip formulation for single machine scheduling, In *FRANCORO/ROADEF*.

[5] J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson (2000), Scheduling aircraft landings - the static case, *Transportation Science* **34**, 180 – 197.

[6] D. Biskup and M. Feldmann (2001), Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers and Op. Research* **28**(8), 787 – 801.

[7] C. Blum (2005), Beam-ACO - hybridizing ant colony optimization with beam search: an application to open-shop scheduling, *Computers and Operations Research* **32**(6), 1565 – 1591.

[8] P. Brucker, S. Heitmann and J. L. Hurink (2003), Flow-shop problems with intermediate buffers, *OR Spectrum* **25**(4), 549 – 574.

[9] P. Brucker, J. Hurink, B. Jurisch and B. Wöstmann (1997), A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics* **76**, 43 – 59.

[10] P. Brucker and O. Thiele (1996), A branch and bound method for the general shop problem with sequence dependent setup-times, *OR Spektrum* **18**, 145 – 161.

[11] K. Bulbul, P. Kaminsky and C. Yano (2001), Preemption in single machine earliness/tardiness scheduling, Submitted for publication.

[12] T. Carchrae and J.C. Beck (2005), Applying machine learning to low knowledge control of optimization algorithms, *Computational Intelligence*, **21**(4), 372 – 387.

[13] J. Crawford (1996), An approach to resource constrained project scheduling, In *Proc. 1996 Artificial Intelligence and Manufacturing Research Planning Workshop.*

[14] E. Danna and L. Perron (2003), Structured vs. unstructured large neighborhood search, In *Proc. CP 2003*, 817 – 821.

[15] U. Dorndorf, E. Pesch and T. Phan-Huy (2001), Solving the open shop scheduling problem, *Journal of Scheduling* **4**, 157 – 174.

[16] B. Fox and M. Ringer (1995), Planning & scheduling benchmarks, URL: www.neosoft.com/˜benchmrx/.

[17] D. Godard, P. Laborie and W. Nuijten (2005), Randomized Large Neighborhood Search for Cumulative Scheduling, In *Proc. ICAPS-05*, 81 – 89.

[18] C. Guéret and C. Prins (1999), A new lower bound for the open-shop problem, *Annals of Operations Research* **92**, 165 – 183.

[19] ILOG (2003), *ILOG OPL Studio 3.7 User's Manual and Reference Manual*, ILOG, S.A.

[20] B. Kara (2002), SMTTP problem library, http://www.bilkent.edu.tr/˜bkara/start.html.

[21] L. Khatib, P. Morris, R. Morris and F. Rossi (2001), Temporal constraint reasoning with preferences, In *Proceedings IJCAI*, 322 – 327.

[22] R. Kolisch and S. Hartmann (2006), Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem: An update, *European Journal of Operational Research* **174**, 23 – 37.

[23] R. Kolisch and A. Sprecher (1996), PSPLIB - A project scheduling problem library, *European Journal of Operational Research* **96**, 205 – 216.

[24] P. Laborie (2003), Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results, *Artificial Intelligence* **143**(2), 151 – 188.

[25] P. Laborie (2005), Complete MCS-Based Search: Application to Resource Constrained Project Scheduling, In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*

(IJCAI-05).

[26] C. Le Pape (1994), Implementation of resource constraints in ILOG Schedule, *Intelligent Systems Engineering* **3**(2), 55 – 66.

[27] T. Morton and D. Pentico (1993), *Heuristic Scheduling Systems*, Wiley.

[28] W. Nuijten (1994), *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, PhD thesis, Eindhoven University of Technology.

[29] W. Nuijten, T. Bousonville, F. Focacci, D. Godard and C. Le Pape (2004), Towards an industrial manufacturing scheduling problem and test bed, In *Proc. PMS*.

[30] I. M Ovacik and R. Uzsoy (1996), Decomposition methods for scheduling semiconductor testing facilities, *International Journal of Flexible Manufacturing Systems* **8**, 357 – 398.

[31] D. Pisinger and S. Ropke (2005), A general heuristic for vehicle routing problems, Technical report, DIKU, University of Copenhagen.

[32] N. Policella, A. Cesta, A. Oddi and S.F. Smith (2004), Generating robust schedules through temporal flexibility, In *Proceedings ICAPS-04*, Whistler, Canada, June.

[33] N. Policella, X. Wang, S.F. Smith and A. Oddi (2005), Exploiting temporal flexibility to obtain high quality schedules, In *Proc. AAAI-2005*.

[34] P. Shaw (1998), Using constraint programming and local search methods to solve vehicle routing problems, In *Proc. CP-98*, 417 – 431.

[35] F Sivrikaya-Serifolu and G. Ulusoy (2004), Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach, *Journal of the OR Society* **55**(5), 504 – 512.

[36] F. Sourd (2006), A reinforced lagrangean relaxation for non-preemptive single machine problem. application to the earliness-tardiness common due date criterion, In *Proc. PMS'06*.

[37] F. Sourd and S. Kedad-Sidhoum (2003), The one machine scheduling with earliness and tardiness penalties, *Journal of Scheduling* **6**, 533 – 549.

[38] F. Sourd and S. Kedad-Sidhoum (2005), An efficient algorithm for the earliness-tardiness scheduling problem, In *Optimization Online*.

[39] R.H. Storer, S.D. Wu, and R. Vaccari (1992), New search spaces for sequencing problems with application to job shop scheduling, *Management Science* **38**(10), 1495 – 1509.

[40] E. Taillard (1993), Benchmarks for basic scheduling problems, *European Journal of Operations Research* **64**, 278 – 285.

[41] P. van Hentenryck, L. Michel, P. Laborie, W. Nuijten and J. Rogerie (1999), Combinatorial optimization in OPL Studio, In *Proc. EPIA 1999*, 1 – 15.

[42] M. Vanhoucke, E. Demeulemeester and W. Herroelen (2001), An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem, *Annals of Operations Research* **102**(1-4), 179 – 196.

[43] T. Yamada and R. Nakano (1992), A genetic algorithm applicable to large-scale job-shop problems, *Proc. Int. Workshop on Parallel Problem Solving from Nature*, 281 – 290.

A Disjunctive Graph for the Job-Shop with Several Robots

Philippe Lacomme, Mohand Larabi
Université Blaise Pascal, LIMOS (UMR CNRS 6158), 63177 Aubière Cedex, France,
{placomme, larabi }@isima.fr

Nikolay Tchernev
IUP « Management et gestion des entreprises », Université d'Auvergne, LIMOS (UMR CNRS 6158),
26, av. Léon Blum, Clermont Ferrand, France (tchernev@isima.fr)

This paper addresses the scheduling problem in a job-shop where the jobs have to be transported between the machines by several transport robots. The problem can be efficiently modeled by a disjunctive graph and any solution can be fully defined by an orientation of the graph. The objective is to determine a schedule of machine and transport operations as well as an assignment of robots to transport operations with minimal makespan.
We present: *(i)* a problem representation using an appropriate disjunctive graph; *(ii)* a solution representation based on 3 vectors consisting of machine disjunctions, transport disjunctions and robots assignments; *(iii)* a new problem-specific properties to define local search algorithms. Computational results are presented for test data arising from Bilge and Uluzoy's benchmark instances enlarged by transportation, empty moving times for each robot. This paper is a step forward definition of an efficient approach arising a job-shop with several robots and it is an attempt to extend the Hurink and Knust proposal dedicated to one single transport robot.

Keywords: scheduling, job-shop, transport.

1. Problem settings and definition

A job-shop scheduling problem with transportation times and several robots can be formulated, in general, as a problem of processing a set of n jobs $J = \{J_1, \ldots, J_n\}$ on a set of m machines $M = \{M_1, \ldots, M_m\}$, transported by a set of r transport robots $R = \{R_1, \ldots R_r\}$. Each job J_i is an ordered set of n_i operations denoted $O_{i1}, O_{i2}, \ldots, O_{i,n_i}$ for which precedence constraints are defined. Each machine can process (without preemption) only one job at one time and each job can be processed by only one machine at the same time. Additionally, transportation times are taken into account considered each time a job changes from one machine to another one. Transportation operation $t^{R_r}_{\mu_{i,k},\mu_{i,k+1}}$ must be considered for robot R_r when machine operation O_{ik} is processed on machine μ_{ik} and machine operation $O_{i,k+1}$ is processed on machine $\mu_{i,k+1}$. These transportation times are job-independent and robot-dependent. Each transportation operation is assumed to be processed by only one transport robot which can handle at most one job at one time. For convenience, $t^{R_r}_{\mu_{i,k},\mu_{i,k+1}}$ is used to denote both a transportation operation and a transportation time.
Also empty moving times $v^{R_r}_{i,j}$ have to be considered while one robot R_r moves from machine M_i to M_j without carrying a job. It is possible to assume, for each robot R_r, that $v^{R}_{i,i} = 0$ and $t^{R_r}_{i,j} \geq v^{R_r}_{i,j}$. As in job-shop problems, we assume that sufficient buffer space exists between machines. This assumption is also stated as an "unlimited input/output buffer capacity". Jobs processed on one machine M_i are assumed to wait until the robot affected to this transport operation is available to do it. No additional time is required to transfer job from machine to the

unlimited output buffer. In a similar way, each machine M_i has an unlimited input buffer where jobs awaited for processing on M_i may be stored.

All data p_{ij}, $t_{i,j}^{R_r}$, $v_{i,j}^{R_r}$ are assumed to be non-negative integers. The objective is to determine a feasible schedule which minimizes the makespan $C_{\max} = \underset{j=1,n}{Max}\{C_j\}$ where C_j denotes the completion time of the last operation O_{j,n_j} of job J_j including starting time of machine and transport operations. Figure 1 provides one solution of the instance provided in appendix 1.

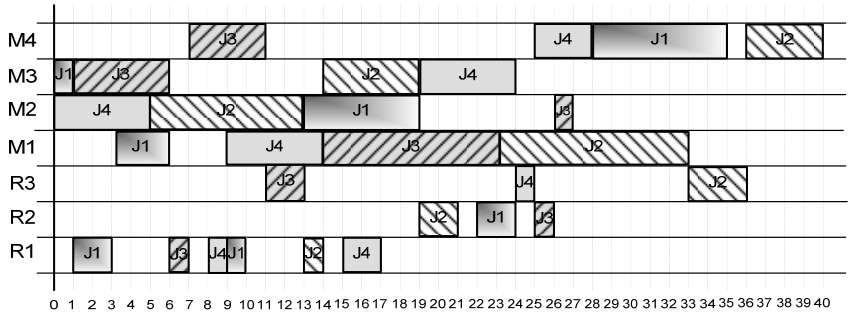


Fig.1. Example of solution where robots are considered as additional machines

The studied problem encompasses both classical job-shop and the job-shop with one single transport robot. Surveys on job-shop are proposed by [1, 2].

Numerous studies tend to include extra constraints in order to take into account transportation constraints in scheduling problems. In [3], authors include delay from the end of processing time on one machine and the earliest due date on the next machine of the same job. This delay is denoted transportation time. Constraints on the number of robots are stressed for the flow-shop by [4] and for the job-shop by [5] and by [6]. Flow-shop with transportation constraints received a considerable amount of attention and a survey is proposed in [7]. One can note a great number of extra constraints in classical job scheduling problems including but not limited to buffers for the flow-shop [8] and set up [9] for the job-shop. To the best of our knowledge [10] and [11] are the last publications on the job-shop including but not limited to the following extra constraints: input/output buffer capacity, no-move ahead trips, FIFO buffer management rule for buffers. A mixed integer linear formulation of the problem is introduced first by [10]. Lastly, Hurink and Knust in 2005 [6] introduced an efficient disjunctive graph for the job-shop with one single robot and derive some specific properties to a proper definition of neighbourhoods in local search procedure.

In this paper we propose a disjunctive graph for the job-shop with several robots and a local search algorithm. This work extends the Hurink and Knust proposal dedicated to one single transport robot. This proposal exploit previous published proposals first initiate by [12] and [13].

2. Disjunctive graph

2.1. Proposal for a non oriented disjunctive graph

In this paragraph the disjunctive graph model for the job-shop problem with transportation time proposed by Hurink and Knust [6] is extended to encompass several robots. Since Hurink's disjunctive graph already deals with all conflicts regarding job-shop machines and scheduling of one robot, we have to incorporate only the assignment of robots to transport operations. Thus, the disjunctive graph $G = (V_M \cup V_t, C \cup D_m \cup D_R)$ consists of: a set of vertices V_M containing all machine operations; a set of vertices V_t is the set of transport operations obtained by an assignment of robot to each transport operation and two dummy nodes 0 and *. The graph consists

also of a set of conjunctions C representing precedence constraints $O_{i,k} \rightarrow t^{R_r}_{\mu_{i,k},\mu_{i,k+1}} \rightarrow O_{i,k+1}$, disjunctions for the machine D_m, disjunctions for the transport D_r and the assignment of robots to transport operations A_R. For each job J_i, $n_i - 1$ transport operations $t^{R_r}_{\mu_{i,k},\mu_{i,k+1}}$ are introduced including precedence $O_{i,k} \rightarrow t^{R_r}_{\mu_{i,k},\mu_{i,k+1}} \rightarrow O_{i,k+1}$.The arcs from machine node to transport node are weighted with the machine operation duration. Edges between machine operations represent disjunctions since no machine operations which have to be processed on the same machine can hold simultaneously. In addition, to the classical set of undirected machine disjunctions D_m (all pairs of operations which have to be processed on the same machine and which are not linked by a directed path), it is necessary to consider the set of undirected robots disjunctions (D_r). Undirected robot disjunctions can not be represented since transport operation assignment (A_R) to one robot is not completed first. Figure 2 represents the non oriented disjunctive graph linked to the data set in the appendix. To solve the scheduling problem it is necessary to turn all undirected arcs in $D_m \cup D_r$ into directed ones, and to assign one robot to each transport operation.

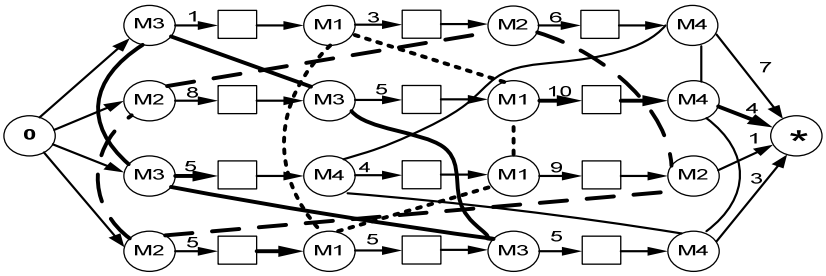


Fig.2. Non oriented disjunctive graph for representation of the problem

2.2. Proposal for an oriented disjunctive graph

Like Knust [13], a fixed machine selection S_m is called Machine Disjunctions, a fixed transport selection S_R is called Transport Disjunction. $\hat{S} = C \cup S_m \cup S_r$ is called a complete selection. The correct definition of S_m, S_R is make using three vectors denoted MS, TS, RA.

It is possible to obtain acyclic graph for the job-shop using the Bierwith's [14] representation first introduced for the job-shop. Bierwith's representation is a "sequence with repetition" and consists in repetition of a job number to model the machine operation.

The machine selection MS is a vector by repetition. Since each job j has n_j machine operations, MS encompasses n_j times the number j. The first occurrence of j in the vector MS refers to the first machine operation of the job, the second occurrence referred to the second one and so on. S_m can be efficiently defined by an ordering loop of the vector MS adding arc in S_m since two operations of different job refer to the same machine.

The transport selection TS is a vector by repetition. Since each job j has $n_j - 1$ transport operations, TS encompasses $n_j - 1$ times the number j. The first occurrence of j refers to the first transport operation of job j from machine $\mu_{j,1}$ to machine $\mu_{j,2}$, the second occurrence of j refers to the second transport operation of job j and so on. This vector permits to define the disjunctive arc between the transport operations. The weight of transport operation arcs depend on the robot assigned to the transport operation. The robot assignment RA is a vector where $RA(i) = p$ defines that the transport operation of job $TS(i) = j$ is assigned to robot p. If the number j in $TS(i)$ is the k^{th} occurrence of j in the vector TS, it refers to the k^{th} transport of job j. If the edge is oriented in the direction

$O_{i,k} \to O_{j,k'}$ it gets the weight p_{ik}. If an arc is added from $t^{R_p}_{\mu_{i,k},\mu_{i,k+1}}$ to $t^{R_p}_{\mu_{j,k'},\mu_{j,k'+1}}$, its gets the weight $t^{R_p}_{\mu_{i,k},\mu_{i,k+1}} + v^{R_p}_{\mu_{i,k+1},\mu_{j,k}}$ and $t^{R_p}_{\mu_{j,k'},\mu_{j,k'+1}} + v^{R_p}_{\mu_{j,k'+1},\mu_{i,k}}$ if its oriented in the other direction as shown in figure 3.

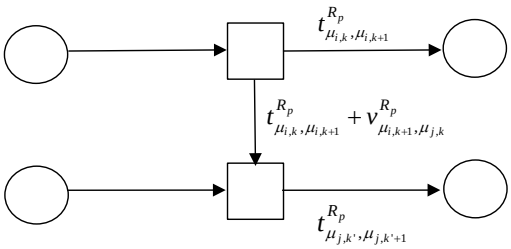


Fig. 3. Disjunction between two transport operations

The union $C \cup S_m \cup S_r \ \hat{S}$ fully describes a solution if the resulting oriented disjunctive graph $G = (V_M \cup V_t, \hat{S})$ is acyclic. A feasible schedule can be constructed by longest path calculation which permits to obtain the earliest starting time of both machine and transport operations and fully defines a semi-active schedule with the $C_{\max}$ given by the length of the longest path from node 0 to *.

The longest path defines the set of operations within any delay cannot be expected between the two dummy nodes ($C_{\max}$). Note that unfortunately there is a great number of selections (MS , TS , RA) which induce an identical oriented disjunctive graph since it is the relative order of operation which is responsible of the disjunctions.

Example. We consider an instance of a job-shop problem with 4 robots, $m = 4$ machines, $n = 4$ jobs, 16 machine operations and 12 transport operations. MS , TS and RA are given in figure 4. MS defines the following order of jobs on machine: Machine 1: Job 1, Job 4; Machine 2: Job 4, Job 2, Job 1, Job 3; Machine 3: Job 1, Job 3, Job 2, Job 4; Machine 4. Job 3, Job 4, Job 1, Job 2. TS and RA define that robot 1 processes job 4 first, job 1 after, job 2 and finally job 4. The first transport of job 1 is assigned to robot 3, the second transport of job 1 is assigned to robot 1 and finally the last transport of job 1 is assigned to robot 2. The corresponding schedule can be found in figure 5 where numbers in bold are earliest starting time of operation. Using a linear formulation of the problem, this solution has been proved to be optimal.

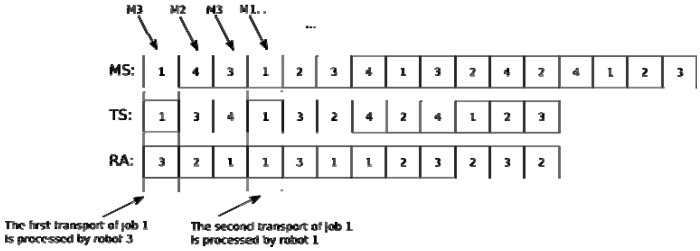


Fig.4. MS, TS and RA for definition of a solution

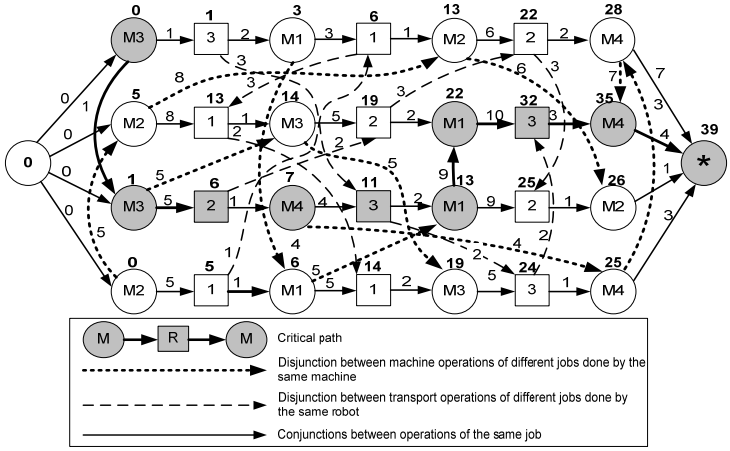


Fig.5. The oriented disjunctive graph for representation of the solution induced by Fig. 4.

2.3. Local search

In this section we introduce some specific properties of problems which can be efficiently included in neighbourhoods for local search approach. In [15] the neighbourhood is based on the permutation of two successive machine operations on the critical path which concerns the same machine. In [16] is presented new restrictions of the Laarhoven's neighbourhood [15]. [17] developed a block approach which is based on machine-block definition. Dell'Amico and Trubian's [18] first introduce a specific machine-operation by exchanging one machine-operation in a block with one machine-operation at the end or at the beginning of the same block.

The search space is the set of all complete selections. At each iteration a neighboured solution is generated by moving an operation on a machine or by moving a transport operation on the robot to another position or by modifying the assignment of a transport operation to another robot. Let us note $P(S)$ the critical path in an acyclic graph. The sequence $u_1, u_2, u_3, ..., u_n$ (where $u_1 = 0$ and u_n the dummy node at the end of the graph) is the sequence of operations (machine operations or transport operations) which defines the critical path. We define also: *(i)* a machine-block is a succession of machine operations in the sequence which are processed consecutively in the critical path; *(ii)* robot-block is a succession of loaded transport operations in the sequence which are processed consecutively in the critical path.

The theorem presented in [6] holds and it is easy to state that to obtain a new neighbourhood solution with a completion time less or equal to this one of the current sequence, either: *(i)* at least two transport operations of a robot-block must be processed in the opposite order; *(ii)* at least two machine operations of a machine-block must be processed in the opposite order.

Let us note that changing the assignment of one robot to one transport operation in the critical path can induce a new solution with completion time less or equal to the current one. Modifications of the graph consist in modifying *MS* or *TS* or *RA* by permutation or by modification of one value in a special rank. Figure 6 represents in grey the positions in MS, TS and RA where modifications are required to obtain a neighbourhood solution with makespan less or equal to 40 (figure 7 gives the graph obtained from the gantt of figure 1).

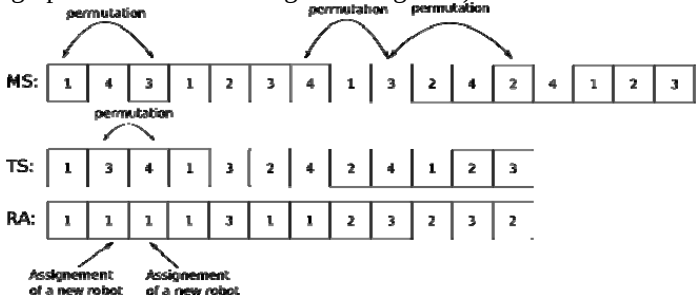


Fig.6. MS, TS and RA with modifications

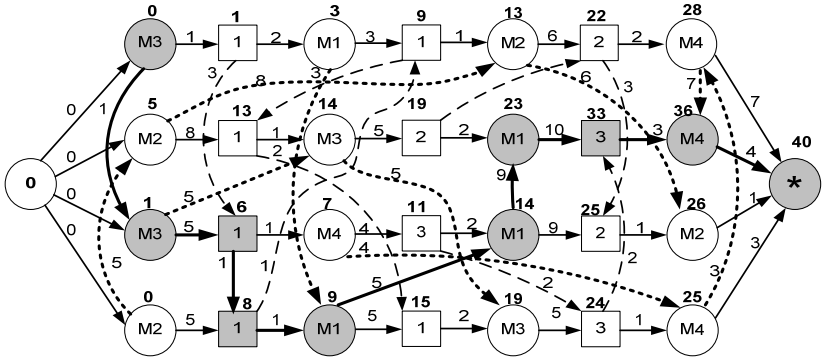


Fig.7. Example of solution with longest path

Algorithm 1. Local search

```
Procedure name: Local_Search
    While (i<>0) and (iter<=nm) do
       j:=P(i)   // j is the father of i on the critical path
       If (job of operation i <> job of operation j) then
          If (i is a machine operation and j is a machine operation) then
           Save_MS := MS
           Swap these two machine operations in MS
           C :=  Evaluate_Graph with disjunctions and affections given by MS , TS , RA
           If ( C  <  C^best ) then
               C^best := C ;     // new best current solution
              i:=P(N+1) // last vertex of the new critical path
           Else
              i:=P(i) // father vertex on the critical path
           End if
          Else
           If (i is a transport operation and j is a transport operation) then
             Save_TS := TS
             Swap these two transport operations in TS
             C :=  Evaluate_Graph with disjunctions and affections given by MS , TS , RA
             If ( C  <  C^best ) then
               C^best := C ;     // new best current solution
              i:=P(N+1) // last vertex of the new critical path
             Else
               i:=P(i) // father vertex on the critical path
             End if
           End if
          Else
           If (i is a transport operation) then
             k:=1; Stop:=false;
             While (k<r) and (stop=false) loop
               If (k=robot of the transport operation i) then
                k:=k+1
               else
                use k as robot of i transport in RA
                C :=  Evaluate_Graph with disjunctions and affections given by MS , TS , RA
                If ( C  <  C^best ) then
                     C^best := C ;  Stop:=True;
                     i:=P(N+1) // last vertex of the new critical path
                Else
                     i:=P(i) // father vertex on the critical path
                End if
               End If
              End Loop
            End If
          End if
          Iter:=Iter+1
    End Do
    Return MS , TS , RA , C^best
End
```

The local search mainspring is given on algorithm 1. It is based on a main loop which iterates along the vertices of the critical path from the last operation to the first one. The algorithm stops when the initial vertex is reached or when the maximal number of iterations (*nm*) is achieved**.** At each iteration, a neighbourhood is generated by permutation of two operations of a block (machine block or transport block) or by an assignment of a new robot to a transport. When a lowest cost neighbourhood solution (the final vertex label of the graph has a lower cost) is obtained, the father vertex is stored in *P(N+1)* and the formula *i:=P(i)* permits to attain the father vertex along the critical path. The beginning and the end of two blocks are identified by two operations of different jobs on the critical path. The swap of these operations permits to generate a neighbourhood solution which is evaluated by Evaluate_Graph. Two situations must be considered: first, the new solution is better than the current one then the new solution becomes the current one ($C^{best} := C$) and the index i is assigned to the last operation of the new critical path ($i := P(N+1)$). Second, the new solution is worst and the index *i* is updated ($i := P(i)$) to access to i father's operation on

the critical path. Note that the modification of robot assignment is investigated whatever the result of a transport-swap operations. In the Algorithm 1, procedure Evaluate_Graph is an efficient implementation of a Bellman like longest path algorithm with cycle detection. We define the following extra variables used in the local search algorithm: *nm* is the maximal number of iterations; C^{best} is the best makepsan; *N* is the number of nodes in the graph; *iter is* the current iteration; C is the makepsan; $Save\_MS$ is the machine selection; $Save\_TS$ is the transport selection and $Save\_RA$ is the robot assignment

3. Numerical experiments

The evaluation is carried out using 20 instances available for download at http://www.isima.fr/lacomme/Job-Shop_2.html. These instances encompass instances with similar robots for both empty and loaded transport and instances with great differences between transportation times of the robots. The local search manages 100 iterations. Trying to highlight the local search efficiency, the experimentations for table 1 were managed from a poor initial solution. By increasing the number of iterations devoted to local search, it is possible to increase the quality of solution. Note the optimal solution is obtained thanks to a time consuming resolution of a linear program.

Table 1
Solution obtained with local search

Nb of robots r	Optimal solution	Initial solution	Initial solution $C^1_{\max}$	Best solution found After 100 iterations	$C^2_{\max}$ After 100 iterations	$C^2_{\max}$ After 1000 iterations
1	40	MS = 1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 TS = 1,1,1,2,2,2,3,3,3,4,4,4 RA = 1,1,1,1,1,1,1,1,1,1,1,1	**85**	MS = 1,1,2,1,1,2,2,3,3,2,3,4,3,4,4,4 TS = 1,1,2,1,2,3,3,2,4,3,4,4 RA = 1,1,1,1,1,1,1, 1,1,1,1,1	54	54
2	39	MS = 1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 TS = 1,1,1,2,2,2,3,3,3,4,4,4 RA = 2,2,2,2,2,2,2,2,2,2,2,2	141	MS = 1,1,2,4,3,3,2,2,2,1,1,4,3,3,4,4 TS = 1,1,3,2,2,4,1,3,3,2,4,4 RA = 2,2,2,1,1,1,2,1,2,1,1,1	47	45
3	39	MS = 1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 TS = 1,1,1,2,2,2,3,3,3,4,4,4 RA = 3,3,3,3,3,3,3,3,3,3,3,3	183	MS = 1,1,2,1,3,2,2,3,1,4,4,4,3,3,2,4 TS = 1,1,1,,2,2,3,4,3,3,2,4,4 RA =1,2,3,1,1,1,12,2,1,1,1	45	45
4	39	MS = 1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4 TS = 1,1,1,2,2,2,3,3,3,4,4,4 RA = 4,4,4,4,4,4,4,4,4,4,4,4	225	MS = 1,1,2,3,1,2,2,3,2,1,4,3,4,3,4,4, TS = 1,1,2,1,2,3,2,3,3,4,4,4 RA =4,4,1,4,1,1,1,1,2,1,1,1	60	45

4. Concluding remarks and future researches

This paper is a step forward generalisation of the disjunctive graph model including several robots. The disjunctive graph is an efficient model of both problem and solution, the longest path permits to derive specific properties to efficient definition of neighbourhoods. The local search we promote takes advantages of both machine-block and transport-block. The problem appears to be hard to solve since an acyclic graph representing of given solution requires a proper coordination of both transport robot and machine disjunction. Our objective consists in defining a memetic based framework. Our research is now directed on:

- a dedicated crossover operation based also on the block;
- generation of initial solution based on Giffler and Thomson based heuristics;
- numerical experiments on instances introduced on Hurink and Knust's proposal and extended with several robots.

Problem characteristics including inconsistent graph push us into focusing first on a proper management of non-feasible selections. This can be achieved in assignment of a specific cost function taking into consideration the infeasibility of the selection.

Appendix 1

Table 2. One instance of job-shop

Job 1	M3 Processing time:	1	M1 Processing time:	3	M2 Processing time:	6	M4 Processing time: 7
Job 2	M2 Processing time:	8	M3 Processing time:	5	M1 Processing time:	10	M4 Processing time: 4
Job 3	M3 Processing time:	5	M4 Processing time:	4	M1 Processing time:	9	M2 Processing time: 1
Job 3	M2 Processing time:	5	M1 Processing time:	5	M3 Processing time:	5	M4 Processing time: 3

Table 2. Empty transportation time

	M1	**M2**	**M3**	**M4**
M1	0	1	1	1
M2	1	0	1	1
M3	1	1	0	1
M4	1	1	1	0

Table 3. Transportation time

	M1	M2	M3	M4
M1	/	1	2	3
M2	1	/	1	2
M3	2	/	/	1
M4	2	/	/	/

/: no transportation time since no transport from M_i to M_j

Note: Robot 2 transportation times are 2 times greater than transportation time of robot 1;
Robot 3 transportation times are 4 times greater than transportation time of robot 1;
Robot 4 transportation times are 8 times greater than transportation time of robot 1;

References

[1] E. Pinson (1995), The job shop scheduling problem: A concise survey and some recent developments, in: P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu (Eds.), *Scheduling Theory and Its Applications*, Wiley, New York, 277 – 293.

[2] A.S. Jain and S. Meeran (1999), Deterministic job-shop scheduling: Past, present and future", *European Journal of Operational Research* **113**(2), 390 – 434.

[3] V.A. Strusevich (1999), A heuristic for the two-machine open-shop scheduling problem with transportation time, *Discrete Applied Mathematics* **93** (2-3), 287 – 304.

[4] J. Hurink and S. Knust (2001), Makespan minimization for flow-shop problems with transportation times and a single robot, *Discrete Applied Mathematics* **112**, 199 – 216.

[5] J. Hurink and S. Knust (2002), A tabu search algorithm for scheduling a single robot in a job-shop environment, *European Journal of Operational Research* **119**(1-2), 181 – 203.

[6] J. Hurink and S. Knust (2005), Tabu search algorithms for job-shop problems with a single transport robot, *European Journal of Operational Research* **162**(1), 99 – 111.

[7] Y. Crama, V. Kats, J. Van Knludert and E. Levner (2000), Cyclic scheduling in robotic flowshop, *Annals of Operations Research* **96**, 97 –124.

[8] P. Brucker, S. Heitmann and J. Hurink (2003), Flow-Shop problems with intermediate buffers, *OR Spectrum* **25**, pp. 549 – 574.

[9] C. Artigues, P. Lopez and P.-D Ayache (2005), Schedule Generation Schemes for the Job-Shop Problem with Sequence-Dependent Setup Times: Dominance Properties and Computational Analysis, *Annals of Operations Research* **138**, 21 – 52.

[10] A. Caumond, P. Lacomme, A. Moukrim and N. Tchernev (2006), A MILP for scheduling problems in an FMS with one vehicle, submitted to *European Journal of Operational Research*.

[11] P. Lacomme, A. Moukrim and N. Tchernev (2005), Simultaneously Job Input Sequencing and Vehicle Dispatching in a Single Vehicle AGVS: a Heuristic Branch and Bound Approach Coupled with a Discrete Events Simulation Model, *International Journal of Production Research* **43**(9), 1911 – 1942.

[12] J. Hurink and S. Knust (2001), List scheduling in a parallel machine environment with precedence constraints and setup times, *OR Letters* **29**, 231 – 239.

[13] S. Knust (1999), Shop-scheduling problems with transportation, Dissertation, Fachbereich Mathematik/Informatik, Universität Osnabrück.

[14] C. Bierwirth (1995), A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spektrum* **17**, 87 – 92.

[15] P.J.M. Van Laarhoven, E.H.L. Aarts and J.K. Lenstra (1992), Jobshop scheduling by simulated annealing, *Operations Research* **40**, 113 – 125.

[16] E. Nowicki and C. Smutnicki (1996), A fast taboo search algorithm for the job-shop problem, *Management Science* **42**(6), 797 – 813.

[17] J. Grabowski, E. Nowicki and S. Zdrzalka (1996), A block approach for single machine scheduling with release dates and due dates. *European Journal of Operations Research* **26**, 278 – 285.

[18] M. Dell'amico and M. Trubian (1993), Applying tabu-search to the job-shop scheduling problem. *Annals of Operations Research* **41**, 231 – 252.

A Compact Optimization Approach for Job-Shop Problems

Giuseppe Lancia, Franca Rinaldi, Paolo Serafini
University of Udine, Dept. of Mathematics and Computer Science, Via delle Scienze 206, Udine, Italy,
{lancia, rinaldi, serafini}@dimi.uniud.it

Abstract: In this paper we propose a time-indexed IP formulation for job-shop scheduling problems. We first introduce a model with variables associated to job scheduling patterns and constraints associated to machine capacities and to job assignments. The exponential number of variables calls for a column generation scheme which is carried out by a dynamic programming procedure. However, the column generation process is time-consuming. Therefore we derive an equivalent compact formulation of the model, whose dual is a flow problem with side constraints. This problem can be solved faster and fits easily into a branch-and-bound scheme.

Keywords: Machine Scheduling, Shop-Floor Scheduling, Compact optimization.

1 Introduction

The job-shop problem has been studied extensively in the last three decades. Although there are good heuristic procedures based on local search (for instance the celebrated shifting bottleneck procedure [1] is a local search with a very large neighborhood), the performance of exact procedures has never been satisfying. The hardness of the job-shop problem, even with respect to other NP-hard problems, has been soon recognized [6].

In this paper we introduce a time-indexed formulation for the general job-shop problem, based on an ILP model with column generation. In our model part of the combinatorial structure is taken care of by the constraints and part is embedded in the matrix columns. By this feature, typical of ILP formulations with column generation, the LP relaxation provides a strong lower bound which has a positive effect in pruning the nodes of the branch-and-bound tree.

However, the time required to reach optimality of the relaxed model by column generation can be very large. Therefore, we have resorted to a compact formulation [4, 7] of the same model. Generally speaking, a compact formulation of a model with an exponential number of constraints is an equivalent formulation in which the exponentially many constraints are replaced by a polynomial number of new constraints (after possibly introducing a polynomial number of new variables). For a model with an exponential number of variables (as in our case), the compact approach can be applied to the dual of the formulation. Clearly the compact formulation yields the same bound as the original LP relaxation, but it can be much faster to solve. Moreover, the typical difficulties of taking care of branching constraints within the column generation scheme disappear in the compact formulation.

The job-shop problem we consider has the objective of minimizing the total job cost, i.e., the sum of the penalties on the job completion times. We do not make any assumption on the type of function measuring the penalty, and earliness/tardiness penalties can be easily accomodated. Furthermore, our model can be extended to deal with some variants of job-shop, like the no-wait job-shop scheduling.

Our formulation can be also adapted to the objective of minimizing the maximum job cost. However, we have observed that it is not as effective as in the previous case and, in this case, it is more convenient to approach the problem by varying the time horizon. Finally, we note that

our model cannot be applied to the open-shop problem, where the job operation sequence is not a-priori fixed.

The positive aspects of time-indexed formulations for scheduling problems have been pointed out in [9]. The idea of using column generation for scheduling problems has already been exploited (see [9, 8]). In [9], a problem with identical machines and several jobs is investigated, and variables are associated to single machine scheduling patterns. In [8], a single machine problem is studied and a time-indexed formulation is turned into a more compact model by exploiting the structure of single machine problems. In the above papers the computational burden of a time-indexed formulation (which can yield a strong lower bound), can be successfully avoided. However, for more complex cases, like the job-shop problem, with many operations and machines, there does not seem to be an easy way to escape from the many constraints of a time-indexed formulation.

The idea of using Lagrangian relaxation to split job precedences and machine constraints has been successfully used in [2] for the case of earliness and tardiness costs. Actually, there are strong similarities between our model and one of the two models presented in [2], although the starting points of the two approaches and the solution techniques are quite different.

2 The job-shop problem

In the job-shop problem, a set M of m machines and a set J of n jobs are given. Every job $j \in J$ consists of a given sequence of $o(j)$ operations $O_1^j \to \cdots \to O_k^j \to \cdots \to O_{o(j)}^j$ and each operation O_k^j has to be processed whithout preemption on the machine $\mu(k,j)$ with a known processing time $q(k,j)$ (where we may admit $q(k,j) = 0$ for some k). A *feasible schedule* of the jobs in J is a set of completion times $t(k,j)$ associated to each operation O_k^j such that: (i) the job precedence relations of the operations are respected and (ii) operations associated to the same machine do not overlap in time.

We assume that, for each job $j \in J$ and $k := 1, \ldots, o(j)$, a function

$$f_{k,j} : R \to R \cup \{+\infty\}, \qquad t(k,j) \mapsto f_{k,j}(t(k,j)) \tag{1}$$

is defined assigning a penalty to each operation completion time. The cost of a feasible schedule can be defined in two alternative ways. In the *total cost model* it is defined as

$$\sum_{j \in J} \sum_{k=1}^{o(j)} f_{k,j}(t(k,j)). \tag{2}$$

In the *max cost model* it is defined as

$$\max_{j \in J} \max_{k=1,\ldots,o(j)} f_{k,j}(t(k,j)). \tag{3}$$

The functions $f_{k,j}$ can model release dates r_{kj} and deadlines d_{kj} for each operation by setting $f_{k,j}(t) = +\infty$ for $t < r(j) + q(k,j)$ and $t > d_{kj}$, respectively. Fixed idle times of the machines can be dealt with by similar techniques.

We consider the problem of finding a feasible schedule of minimum cost. In the sequel we assume that all the data are integer numbers. As a consequence, we may restrict our attention to integral completion times $t(k,j)$.

3 A time-indexed formulation

We model the job-shop problem with an objective function of the form (2) or (3) as an integer linear programming problem.

In a discrete-time framework, we denote by $[a, b]$ the set of integers $\{z \in Z : a \leq z \leq b\}$ and call it *time interval*.

A *scheduling pattern* (or simply a pattern) p for the job $j \in J$ is a sequence of $o(j)$ time intervals $[s(k), t(k)]$, $1 \leq k \leq o(j)$ with $t(k-1) \leq s(k)$ for each $2 \leq k \leq o(j)$ and $t(k) - s(k) = q(k, j)$ for each k. In the total cost model the pattern cost is given by

$$c(p) = \sum_{k=1}^{o(j)} f_{kj}(t(k)) \tag{4}$$

and in the max cost model the pattern cost is given by

$$c'(p) = \max_{k=1\ldots o(j)} f_{kj}(t(k)). \tag{5}$$

In the frequent case in which only the last operation of each job contributes to the objective function, i.e. $f_{kj}(t) = 0$ for $k < o(j)$, we have $c(p) = c'(p)$.

Note that a pattern is a particular schedule of the job operations. If p is a pattern for job j, we denote by $t(p)$ the completion time of the last operation of the pattern p, i.e. $t(p) := t(o(j))$.

Let $\overline{T}$ be a value for the time horizon. We assume that $\overline{T}$ is sufficiently large to contain an optimal schedule. We comment on this assumption in the next section. Let us denote by P^j the set of patterns for job j with $t(p) \leq \overline{T}$.

We associate to each pattern p in P^j an $m\,\overline{T}$ dimensional $\{0, 1\}$–vector a^p with m fields of length $\overline{T}$, one for each machine $h \in M$. The t-th entry of the h-th field $a^p_{h,t}$ is 1 if and only if the operation of the job which must be executed by machine h is processed in the time slot $[t-1, t]$. In other words, for each k and for each t, the component $a^p_{\mu(k,j),t}$ is 1 if and only if $s(k)+1 \leq t \leq t(k)$.

Now we associate a binary variable x_p to each pattern p of P^j, with the meaning that $x_p = 1$ if and only if the job j is scheduled according to the pattern p. Then the job-shop problem with min total cost objective may be formulated as

Problem IPT

$$\min \quad \sum_{j \in J} \sum_{p \in P^j} c(p)\, x_p$$

$$\sum_{p \in P^j} x_p = 1 \qquad j \in J \tag{6}$$

$$\sum_{j \in J} \sum_{p \in P^j} a^p_{h,t}\, x_p \leq 1 \qquad h \in M, \quad t = 1, \ldots \overline{T} \tag{7}$$

$$x_p \in \{0, 1\} \qquad p \in P^j, \quad j \in J. \tag{8}$$

Constraints (6) state that each job has to be scheduled according to exactly one pattern and are thus called *assignment constraints*. Conditions (7) guarantee that each machine processes no more than one job at a time and are called *machine constraints*. Finally, we have the binary conditions (8) on the variables. The integrality relaxation of Problem IPT will be denoted as Problem $\overline{\text{IPT}}$. The

job-shop problem with min max cost objective may be similarly formulated by adding to IPT the constraints

$$\sum_{p \in P^j} c'(p)\, x_p \leq z \qquad j \in J \tag{9}$$

and minimizing z. Let us call this problem IPM and its integrality relaxation $\overline{\text{IPM}}$.

Although problems IPT and IPM differ only in the objective function and the pattern structure is the same, it turns out that the lower bound provided by $\overline{\text{IPM}}$ to IPM is very poor, while the one given by $\overline{\text{IPT}}$ to IPT is strong. For this reason we will tackle Problem IPM in an indirect way by setting particular values, in a binary search strategy, for the time horizon $\overline{T}$ in Problem IPT.

Problem IPT contains a constraint matrix having $|J| + m\,\overline{T}$ rows and a huge number of columns, one for each feasible pattern of each job. In order to solve IPT, one has to resort to a column generation approach.

4 The pricing procedure

Let us denote by $u(j)$ the dual variable corresponding to the j-th assignment constraint and by $v(h,t) \leq 0$ the dual variable corresponding to machine h and the time slot $[t-1,t]$, $t \in T(h)$. Moreover for a job j, operation $k \in \{1,\ldots,o(j)\}$ and pattern $p \in P^j$, let us denote by $s(k,j,p)$ the starting time of operation k within pattern p. Then the dual of $\overline{\text{IPT}}$ is :

$$\begin{aligned}
\max \quad & \sum_{j \in J} u(j) + \sum_{h \in M} \sum_{t=1}^{\overline{T}} v(h,t) \\
& u(j) - \sum_{k=1}^{o(j)} \sum_{t=s(k,j,p)+1}^{s(k,j,p)+q(k,j)} (-v(\mu(k,j),t)) \leq c(p) \qquad p \in P^j, \quad j \in J \\
& v(h,t) \leq 0.
\end{aligned}$$

If $\hat{u}$ and $\hat{v}$ denote the optimal dual variables of the master problem, then checking dual feasibility is carried by solving the following n independent problems, one for each job j, and checking nonnegativity of the result:

$$-\hat{u}(j) + \min_{p \in P^j} c(p) + \sum_{k=1}^{o(j)} \sum_{t=s(k,j,p)+1}^{s(k,j,p)+q(k,j)} (-\hat{v}(\mu(k,j),t)). \tag{10}$$

The pricing problem (10) may be solved independently for each job j by a forward dynamic programming procedure. We first note that, apart from the constant $\hat{u}(j)$, the reduced cost of a pattern $p \in P^j$ is given by its original cost $c(p)$ augmented by the sum of the dual variables associated to the time slots when operations are processed according to p. So we may interpret each value $-\hat{v}(h,t)$ as an additional cost on the use of machine h in the time interval $[t-1,t]$. Let $\tilde{v}(k,t) := \sum_{\tau=t-q(k)+1}^{t} (-\hat{v}(\mu(k),\tau))$ (here and in the sequel we occasionally simplify the notation by dropping the dependence on j).

In order to solve (10), we define labels $V(k,t)$, $k := 1,\ldots,o(j)$, $t \geq l(k)$, with $l(k) := \sum_{h=1}^{k} q(h)$. Each $V(k,t)$ represents the minimum reduced cost of a pattern consisting of the first k operations and completing the k-th operation *within* t. Let us conventionally set $V(0,t) := 0$ for each t and $V(k,t) := +\infty$ for each k and $0 < t < l(k)$.

The labels $V(k,t)$ can be recursively computed, for $k := 1,\ldots,o(j)$ and $l(k) \leq t$, as

$$V(k,t) = \min\{V(k,t-1), V(k-1,t-q(k)) + \tilde{v}(k,t) + f_{kj}(t)\} \tag{11}$$

where the two terms in the above expression represent the minimum reduced cost of patterns which complete the k-th operation before t and exactly at time t, respectively. As a consequence, for any time t, the minimum reduced cost of a pattern $p \in P^j$ with $t(p) \leq t$ is given by $V(o(j),t) - \hat{u}(j)$.

From the computational point of view, it is clearly convenient to work with values of $\overline{T}$ as small as possible. On the other side, if the optimal solutions have completion times larger than $\overline{T}$, they cannot be produced if $\overline{T}$ is chosen too small. In order to be sure that optimal solutions are not lost with the chosen time horizon $\overline{T}$, the following result (whose proof is omitted) can be useful. Given the current optimal solution $\overline{x}$ of the relaxation $\overline{\text{IPT}}$, let $T(\overline{x}) := \min\{\tau : \hat{v}(k,t) = 0 \text{ for each } k \in M, t > \tau\}$. Note that by the complementarity conditions, $T(\overline{x}) \leq \max\{t(p) : x_p > 0\}$.

Proposition: *Let us assume that the functions $f_{kj}(t)$ are non decreasing for $t \geq T(\overline{x})$. If, for each $j \in J$, $T(\overline{x}) + \sum_{k=1}^{o(j)} q(k) \leq \overline{T}$, then $\overline{x}$ is an optimal solution of $\overline{IPT}$.*

We have found that a large number of columns need to be generated before reaching optimality in $\overline{\text{IPT}}$. This heavily affects the computing times. In order to overcome this difficulty we have reformulated the dual of $\overline{\text{IPT}}$ in a compact way with size polynomial in the number of jobs and machines, and pseudopolynomial in the processing times.

5 A compact formulation

The dynamic programming recursion formula (11) allows rewriting the dual problem of $\overline{\text{IPT}}$ as:

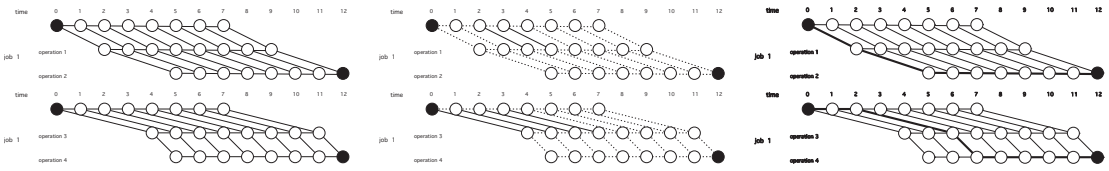
$$
\begin{aligned}
\max \quad & \sum_{j \in J} u(j) + \sum_{h \in M} \sum_{t=1}^{\overline{T}} v(h,t) \\
& V(k,t,j) - V(k,t-1,j) \leq 0, \\
& \qquad \text{for} \quad \sum_{h=1}^{k} q(h,j) + 1 \leq t \leq \overline{T} - \sum_{h=k+1}^{o(j)} q(h,j),\ 1 \leq k \leq o(j),\ j \in J \\
& V(k,t,j) - V(k-1,t-q(k,j),j) \leq \sum_{\tau=t-q(k)+1}^{t} (-v(\mu(k),\tau)) + f_{kj}(t), \\
& \qquad \text{for} \quad \sum_{h=1}^{k} q(h,j) \leq t \leq \overline{T} - \sum_{h=k+1}^{o(j)} q(h,j),\ 1 \leq k \leq o(j),\ j \in J \\
& V(o(j),\hat{T},j) - V(0,0,j) \geq u(j), \qquad \text{for} \quad j \in J \\
& v(h,t) \leq 0
\end{aligned}
\tag{12}
$$

The dual of (12) is a special flow problem on a network with nodes labeled as

$$
\left\{ (j,k,t) \ : \ j \in J,\ 0 \leq k \leq o(j),\ \sum_{h=1}^{k} q(h,j) \leq t \leq \overline{T} - \sum_{h=k+1}^{o(j)} q(h,j) \right\}
$$

and arcs of type 0

$$
\left\{ (j,k,t-1) \rightarrow (j,k,t) \ : \ j \in J,\ 1 \leq k \leq o(j),\ \sum_{h=1}^{k} q(h,j) + 1 \leq t \leq \overline{T} - \sum_{h=k+1}^{o(j)} q(h,j) \right\}
$$



(a) (b) (c)

Figure 1:

with arc variables $\xi^0_{j,k,t}$ and arcs of type 1

$$\left\{ (j,k-1,t-q(k)) \to (j,k,t) \ : \ j \in J,\ 1 \le k \le o(j),\ \sum_{h=1}^{k} q(h,j) \le t \le \overline{T} - \sum_{h=k+1}^{o(j)} q(,j) \right\}$$

with arc variables $\xi^1_{j,k,t}$. The network is disconnected and each connected components is associated to a job. The nodes $(j,0,0) =: s_j$ are sources and the nodes $(j,o(j),\hat{T}) =: d_j$ are sinks. The constraints are as follows: there is flow conservation on all nodes except the sources and the sinks. Each source s_j must send a unit flow to d_j. Furthermore there are special machine constraints across the arcs (and across the connected components): for each machine h and each time t

$$\sum \left\{ \xi^1_{j,k,\tau} \ : \ \tau - q(k) + 1 \le t \le \tau \ ,\ h = \mu(k,j) \right\} \le 1$$

The objective is

$$\min \qquad \sum_{j,k,t} f_{kj}(t)\, \xi^1_{j,k,t}$$

Let us call JSNET this model. An integral flow of JSNET is necessarily a 0-1 flow corresponding to a set of paths from the sources to the sinks and a schedule $t(k,j)$ is immediately derived as $t(k,j) = \left\{ t : \xi^1_{j,k,t} = 1 \right\}$. Moreover, an integral flow corresponds to a set of patterns (one for each job).

A non integral flow can be decomposed into a finite number of source-sink paths and hence an optimal set of patterns for $\overline{\mathrm{IPT}}$ is immediately available.

As an example of JSNET consider an instance with two jobs and four operations with processing times 2, 3, 4 and 1 respectively. The first job consists of operations 1 and 2 and the second job consists of operations 3 and 4. Let us suppose that operations 1 and 3 require one machine, while operations 2 and 4 require a second machine. By fixing a time horizon $\overline{T} = 12$ we build the network in Figure 1(a), where all arcs are directed to the right and down. The black nodes are the sources and the sinks. In Figure 1(b) the non dashed arcs are those entering the machine constraints associated to the first machine and time 4. In Figure 1(c) the optimal solution is shown for the objective function $f_{k,j}(t) = t$, for all k, j.

6 Computing a schedule from a fractional solution

An optimal fractional flow of JSNET can be used to generate feasible solutions of the problem by a randomized rounding procedure. First, for each job j, we recursively compute a path from s_j to d_j as follows: once a node (j,k,t) has been reached, the next node of the path is selected with probability proportional to the flow values of the two arcs exiting from (j,k,t). This is the same as

decomposing the flow in paths and choosing randomly a path with probability proportional to its flow value. Assume that a path, i.e., a scheduling pattern $[s(k,j), t(k,j)]$, $1 \leq k \leq o(j)$, has been computed for all jobs j. Then, we may determine a permutation of the jobs on each machine m by sorting the intervals $[s(k,j), t(k,j)]$, $\mu(k,j) = m$, with respect to the $s(k,j)$ values. Operations with overlapping intervals are randomly switched according to a specific rule. Once the precedence graph is defined, if the functions $f(k,j)$ are regular (i.e. non decreasing functions of the completion times), the cost of the corresponding schedule can be computed as a function of the longest paths on the graph. When the functions $f(k,j)$ are not regular, as in the case of earliness costs, the cost of the schedule can be determined by a LP procedure.

Since the procedure is randomized, we can generate many solutions and apply to the best ones a local search heuristic based on 2–OPT moves. This composite procedure provides good feasible solutions.

7 Preliminary computational results

We first solved model IPT by implementing a branch-and-price algorithm in C, using CPLEX 8.1 as the LP solver. After each column generation, the algorithm also launches a local search procedure, obtaining feasible solutions starting from the current fractional solution. As the computational results of this approach were not satisfying (the approach is too slow) we developed the compact model. However, we do not have yet an efficient code for the integral JSNET. Currently, we solve it through the MPL modeling system which calls CPLEX 8.1 as a MIP solver with the option Primal Net. Feasible solutions are also obtained from the optimal fractional solution through the local search described in the previous section.

In order to compare the behaviour of IPT vs integral JSNET we ran two sets of flow-shop instances with 10 jobs and 3 and 5 machines, respectively, with objective function the sum of job completion times. Each set contains 10 instances with processing times randomly generated in the range [3, 10] according to [10]. We fixed a time limit of 600 seconds. Both procedures found an optimal solution for all the instances with 3 machines although only the JSNET proved the optimality. The average gap $(\mathrm{OPT} - \mathrm{LB})/\mathrm{OPT}$ between the optimum value and the LP lower bound was 2.9%. No instance with 10 jobs was solved to optimality and the average gap with respect to the local search upper bounds was 4.2% at the root node and 3.9% at the time limit.

We also applied the compact formulation (without the heuristic) to the job shop instances with earliness/tardiness costs, 2 machines, 10 and 15 jobs proposed in [2]. The results are reported in Table 1, where LB is the JSNET value at the root node of the b&b tree, UB is the best found solution and the * means that the upper bound is proved optimal, the CPU time is in seconds and the GAP is expressed as percentage and is $(\mathrm{UP} - \mathrm{LB})/\mathrm{OPT}$. Note that for most instances with 10 jobs we obtained a guaranteed optimal solution.

We have also tested the makespan minimization by changing the time horizon in a binary search fashion. The objective function was set to zero and the time horizon was changed according to feasibility or infeasibility of the JSNET model (without integrality requirement). The 6x6x6 instance from [5] was solved optimally providing an optimal value of the JSNET model of 55, which is also the optimal (integer) value of the instance.

8 Conclusions

The relaxation of the model proposed in this paper yields a strong bound for a branch-and-bound search. However, being a time-indexed formulation, the LP problem to be solved can be quite large

Instance	LB	UB	CPU	GAP
I-10-2-tight-equal-1	460	462*	38	0,43
I-10-2-tight-equal-2	437	491	600	10,99
I-10-2-loose-equal-1	218	225*	36	3,11
I-10-2-loose-equal-2	313	320*	217	2,19
I-10-2-tight-tard-1	178	180*	28	1,11
I-10-2-tight-tard-2	144	146*	90	1,37
I-10-2-loose-tard-1	413	416*	108	0,72
I-10-2-loose-tard-2	136	138*	140	1,45
I-15-2-loose-equal-1	1028	1066	600	3,56
I-15-2-loose-equal-2	491	505	600	2,77
I-15-2-tight-tard-1	783	896	600	12,61
I-15-2-tight-tard-2	901	906*	380	0,55
I-15-2-loose-tard-1	646	686	600	5,83
I-15-2-loose-tard-2	279	318	600	12,26

Table 1:

and its solution can be quite time-consuming. We are currently investigating on how to speed up this phase of the computation.

References

[1] J. Adams, E. Balas and D. Zawack (1988), The shifting bottleneck procedure for job-shop scheduling, *Management Science*, **34**, 391-401.

[2] P. Baptiste, M. Flamini and F. Sourd (forthcoming), Lagrangian bounds for just-in-time job-shop scheduling, *Computers & Operations Research*, doi: 10.1016/j.cor.2006.05.009.

[3] P. Brucker (1995), *Scheduling Algorithms*, Springer, Berlin.

[4] R. D. Carr and G. Lancia, (2002), Compact vs Exponential-Size LP Relaxations, *Operations Research Letters*, **30(1)**, 57–65.

[5] H. Fisher and G.L. Thompson, (1963), Probabilistic learning combinations of local job-shop scheduling rules, in *Industrial Scheduling*, J.F. Muth and G.L. Thompson, eds., Prentice-Hall, Englewood Cliffs, NJ, USA.

[6] B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan (1977), Job-shop scheduling by implicit enumeration, *Management Science*,**34**, 441-450.

[7] K. Martin (1991) Using Separation Algorithms to Generate Mixed Integer Model Reformulations, *Operations Research Letters* **10**, 119 – 128.

[8] J.M. Van den Akker, H. Hoogeveen and S. Van de Velde, Parallel machine scheduling by column generation, *Operations research* **47**, 862–872.

[9] J.M. Van den Akker, C.A.J. Huskens and M.W.P. Savelsbergh (2000), Time-Indexed formulations for machine scheduling problems: column generation, *Informs Journal on Computing* **12**, 111 – 124.

[10] J.P. Watson, L. Barbulescu, A.E. Howe, and L.D. Whitley (1999), Algorithm Performance and Problem Structure for Flow-shop Scheduling, *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 688 – 695.

Minimizing Makespan with Multiple Orders per Job in Mixed Flowshops

Jeffrey D. Laub
P.O. Box 13199, Chandler, AZ 85248, USA, Jeff.Laub@asu.edu
John W. Fowler, Ahmet B. Keha
Arizona State University, PO Box 875906 Tempe, AZ 85287-5906, USA,
{John.Fowler, Ahmet.Keha }@asu.edu

We investigate a new scheduling problem, multiple orders per job (MOJ), to optimize the makespan in a 3-machine flowshop consisting of two types of machines: item-processing and lot-processing. MOJ problems are restricted in the number of jobs formed from orders as well as a job capacity limit. Job processing time for an item-processing machine is proportional to the size of the job, while job processing time for a lot-processing machine is not. We extend prior results by characterizing the opposing effects of item/lot machines on optimal job formation and job scheduling in flowshops that contain both types of machines. Job domination in this context is defined, and precise conditions are provided to determine when lower bounds are equivalent to smaller problems.

Keywords: Scheduling, flowshops, multiple-orders-per-job, makespan, semiconductor manufacturing.

1. Introduction

The multiple orders per job (MOJ) problem domain evolved initially from the semiconductor manufacturing industry. In this highly competitive industry, manufacturers must continually improve their processes and equipment. The shift to larger, 300-mm diameter silicon wafers increases the number of good die per wafer by almost 125%, and therefore reduces the number of wafers required to fill an individual order. Transporting these wafers within the production facility to the proper tool groups is accomplished with Front-Opening Pods (FOUPs). These transporters provide a clean atmosphere to prevent wafer surface contamination, and eliminate the risk of damage from personnel physically moving these heavy materials. Although tracking and facilty routing encourage avoiding the separation of wafers in one order, using a complete FOUP for each order is not efficient. Efficiency and equipment utilization can be improved by placing multiple orders within each transporter without splitting any order. This MOJ problem is complicated by the additional constraint of a limited number of transporters and a limited transport capacity.

The multiple-orders-per-job problem for flowshops is depicted in Figure 1. We assume a set of H orders $O=\{o_k\}$ are available for processing and scheduling on sequential machines at time zero. Each order has size s_k, representing the number of items (wafers) in the order, and P is the sum of all order sizes. To accommodate different machine speeds, the unit processing time, c_i, is dependent on machine i, but not on the order. This scenario commonly holds for semiconductor manufacturing equipment where the processing time is wafer-based. For convenience, we also define processing time ratios $\rho_i = c_{i+1}/c_i$.

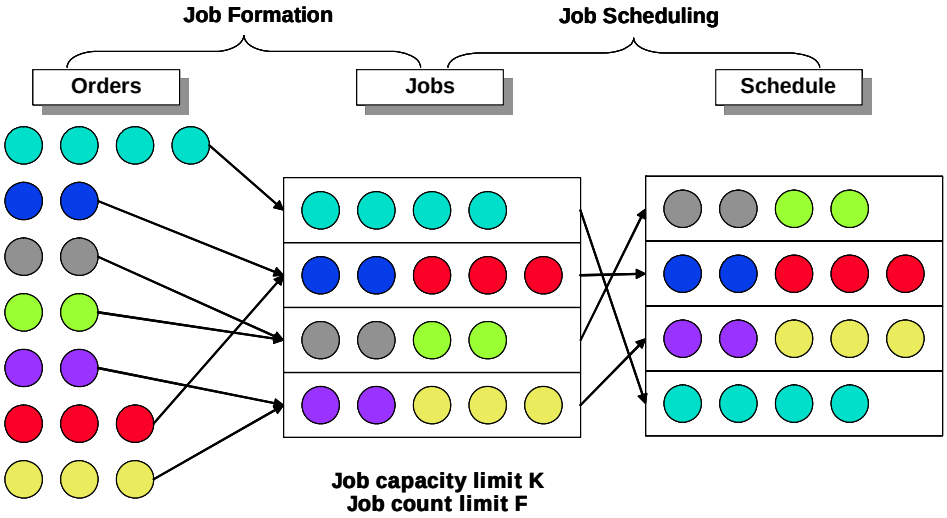


Figure 1. Multiple-Orders-per-Job Problems

Each order must be assigned to exactly one of *F* non-preemptible jobs, and multiple orders can be assigned to a single job as long as the sum of order sizes does not exceed the job size capacity *K*. If the size of an order exceeds *K*, then we assume the order is first divided into separate orders, each of size *K* or less. Each of the resulting orders is therefore wholly contained in a single job. The job processing time depends on the type of machine. On an item-processing machine, denoted "moj(item)", the job processing time is the sum of the processing times for all items in all orders in the job. On a lot-processing machine, denoted "moj(lot)", the processing time is independent of the job composition. The objective is to assign the orders to jobs and schedule the jobs to minimize the makespan for the entire set of orders.

Laub, *et al.* (2005) addressed the two-machine item-processing MOJ makespan problem by calculating a lower bound based on a relaxed problem equivalent to lot streaming with job capacity limits. The calculation of the lower bound was accompanied by equations for optimal job sizes as a function of machine unit processing times, job capacity, and the limit on the number of jobs constructed. The paper then defines an algorithm (which we reference here as "MOJF") that uses these job sizes in a modified version of the bin packing algorithm called "First or Closest Fit Decreasing" (FCFD). Orders are assigned to jobs using the optimal job sizes as "target" bin sizes. The FCFD heuristic attempts to assign orders to jobs to match the target job sizes as closely as possible, but is allowed to exceed these targets as long as the maximum job capacity *K* is not exceeded. The heuristic is quite fast, and produces solutions within a small percentage of the lower bound. Laub, *et al*. (2006 and 2006a) performed the same technique for flowshops with three item processing machines. Their work produced algorithms for precise calculation of the lower bounds as well as further heuristics, as the equations for three machines became much more complicated.

The current paper addresses the multiple orders per job problem for flowshops that contain a mix of both item processing and lot processing machines. For these mixed MOJ machine problems, we use the extended notation *Fm|moj(x,y,...)|Cmax*, where *x, y,…* represent either item-processing or lot-processing machines. For example, *F2|moj(item,lot)|Cmax* represents a two-machine flowshop where the first machine uses item processing and the second machine uses lot processing. Alternatively, for extended configurations, we use abbreviations *I* and *L* for item processing and lot processing, so that, for example, the *F4|moj*(*item*,*item*,*lot*,*lot*)*|Cmax* problem is abbreviated *F4|moj(IILL)|Cmax*, or simply *IILL*. Because we use corresponding lot streaming problems to determine lower bounds for the MOJ problems, we use a similar notation for these, e.g. *LS(IILL)*.

In all cases, a vector $(c_1, c_2,..., c_m)$ defines the times it takes each machine to process a single item. All items in orders are considered equivalent in this work. Let P represent the total number of items in all orders, so that on an item processing machine M_i, the total processing time for all orders is Pc_i and on a lot processing machine, the total processing time is nc_i, where n is the number of jobs constructed. The required processing time for a single job on an item processing machine is $x_j c_i$, where x_j is the number of items in the job (the job size), and the required time for a job on a lot processing machine is simply d, regardless of the job size. In problems involving a limit on job capacity, we use K to represent the job capacity, or the maximum number of items in a job.

2. Three-Machine Mixed Flowshops

We first investigate the three-machine problem with a single lot-processing machine. The position of the lot-processing machine in a three machine flowshop is critical in its effect on the optimal job sizes.

To address this mixed-mode MOJ problem, we first investigate the optimal job sizes for the problem when K is large enough to require no capacitation for an optimal solution. Resulting equations can then be adapted for the capacitated version. The final optimal job sizes are then suitable for use as target job sizes in the First or Closest Fit Decreasing (FCFD) algorithm, and the corresponding makespan is used as the lower bound for the MOJ problems. For convenience when addressing these problems with two item-processing machines, we use d to represent the unit processing time of the lot machine, and c_1 and c_2 as the unit processing times for the two item machines, regardless of their position relative to the lot-processing machine in the flowshop. We use ρ to represent the ratio of processing times between the two item machines (c_2/c_1). As the results are dependent on whether the item processing time ratio ρ is greater than 1, less than 1, or equal to 1, these cases are presented in separate sections.

In these problems, it is not unwarranted to expect the two item machines to compete with the lot machine in terms of optimal job sizes. Potts and Baker (1989) showed that for an uncapacitated flowshop with two item processing machines, there is an optimal solution to the lot streaming problem that uses all available jobs, and is geometric in the optimal job sizes. For a flowshop with lot processing machines, however, an optimal solution will use the minimum number of jobs possible. For the mixed cases, the question is when and how does the speed of the lot processing machine affect the geometric pattern of optimal job sizes for the item machines.

We say that *the item-processing machines dominate the lot processing machine* if the optimal solution to a mixed 3-machine MOJ problem is equivalent, in terms of job sizes, to the corresponding 2-machine problem instance without the lot processing machine. We say that *the lot-processing machine dominates the item-processing machines* if the optimal solution to the problem is equivalent, in terms of job sizes, to the corresponding single-machine problem without the item-processing machines, i.e., all orders are placed in a single job.

We expect that the item processing machines would dominate the lot processing machine for small values of d relative to c_1 and c_2, and the lot-processing machine would dominate the item-processing machines for large values of d. This is, in fact, the case, and the following theorems define these intervals of values for d precisely for each of the three configurations, IIL, LII, and ILI.

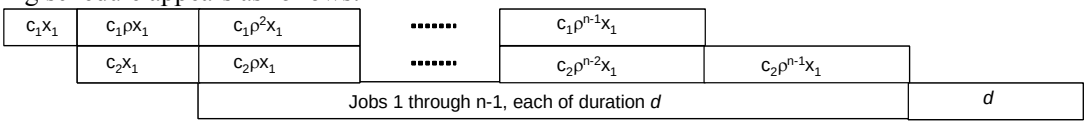
2.1. Uncapacitated IIL

For the following theorems, we relax the problem by assuming that the total amount of processing material is infinitely divisible. The condition for the lot machine dominating the item machines does not depend on which item machine is faster. Theorem 1 states the case for a dominating lot machine when machine M_1 is faster than M_2, but the result for the opposite relationship is the same.

Theorem 1 (Lot Machine Dominance). For the 3-machine lot streaming problem where machines M_1 and M_2 are item-processing machines and machine M_3 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1>1$ and $a_z=P(c_1+c_2)$. Then $d \geq a_z$ if and only if the optimal job configuration is to place all processing in one job. Furthermore, the makespan is $P(c_1+c_2)+d$.

Corollary 1: For any *F3 | moj(mixed) | Cmax* problem with two item-processing machines and one lot-processing machine, the lot machine dominates the item machines whenever $d \geq P(c_1+c_2)$, where the c_i are the unit processing times of the two item machines, regardless of their positions relative to the lot machine.

On the other hand, to characterize the conditions for item machine dominance, we must consider which item machine is the faster one. For the case when $\rho>1$, the optimal item machine dominating schedule appears as follows:



Theorem 2 (Item Machine Dominance for IIL and $\rho>1$). For the uncapacitated 3-machine MOJ problem where machines M_1 and M_2 are item-processing machines and machine M_3 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1>1$ and $a_1 = \dfrac{Pc_2\rho\left(\rho^{n-1}-1\right)}{(n-1)\left(\rho^{n}-1\right)}$. Then the item-processing machines dominate the lot-processing machine if and only if $d \leq a_1$. Furthermore, the makespan is $c_1 P \dfrac{\rho-1}{\rho^n-1} + c_2 P + d$.

We can now characterize the precise values of d that bound the dominating configurations for $\rho>1$, as depicted below.

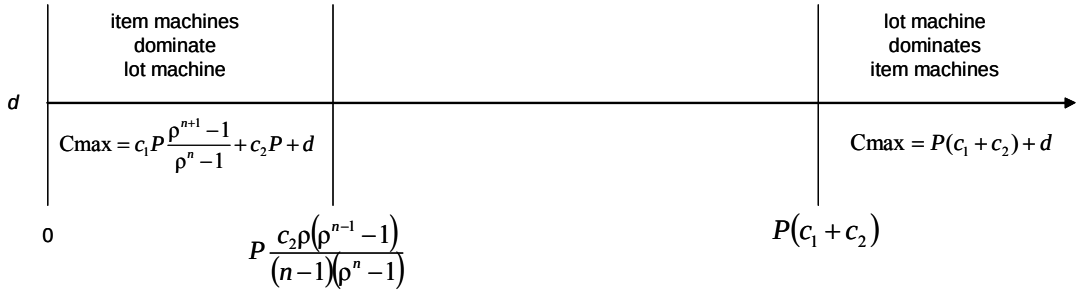
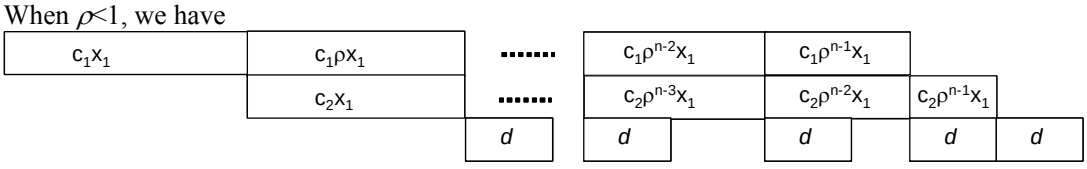


Figure 2. Machine Dominance for IIL, $\rho>1$

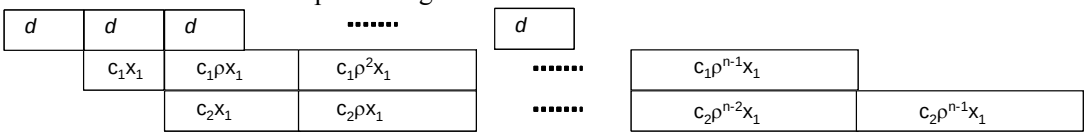
When $\rho<1$, we have



Theorem 3 (Item Machine Dominance for IIL and $\rho<1$). For the uncapacitated 3-machine MOJ problem where machines M_1 and M_2 are item-processing machines and machine M_3 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1<1$ and $a_1 = c_2\rho^{n-1}P\dfrac{\rho-1}{\rho^n-1}$. Then the item-processing machines dominate the lot-processing machine if and only if $d \le a_1$. Furthermore, the makespan is $c_1P\dfrac{\rho-1}{\rho^n-1}+c_2P+d$ (The case for $\rho=1$ is addressed in Section 2.4).

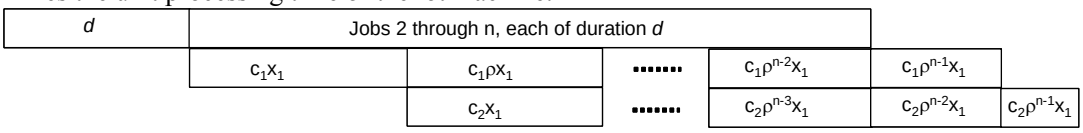
2.2. Uncapacitated LII

When $\rho > 1$ and the item machines are dominant, we have the following optimal schedule that maximizes the allowed unit processing time of the lot machine.



Theorem 4 (Item Machine Dominance for LII and $\rho>1$). For the uncapacitated 3-machine MOJ problem where machines M_2 and M_3 are item-processing machines and machine M_1 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1>1$ and $a_1 = c_1P\dfrac{\rho-1}{\rho^n-1}$. Then the item-processing machines dominate the lot-processing machine if and only if $d \le a_1$. Furthermore, the makespan is $d + c_1P\dfrac{\rho-1}{\rho^n-1}+c_2P$.

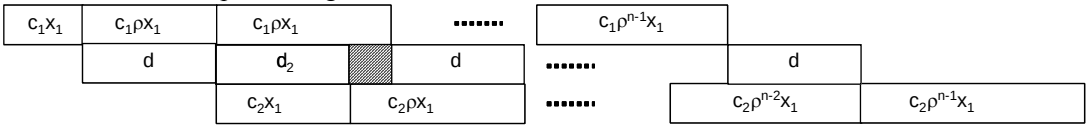
When $\rho < 1$ and item machines are dominant, we have the following optimal schedule that maximizes the unit processing time of the lot machine.



Theorem 5 (Item Machine Dominance for LII and $\rho<1$). For the uncapacitated 3-machine MOJ problem where machines M_2 and M_3 are item-processing machines and machine M_1 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1<1$ and $a_1 = \dfrac{Pc_1\left(\rho^{n-1}-1\right)}{(n-1)\left(\rho^n-1\right)}$. Then the item-processing machines dominate the lot-processing machine if and only if $d \le a_1$. Furthermore, the makespan is $d + c_1P\dfrac{\rho-1}{\rho^n-1}+c_2P$.

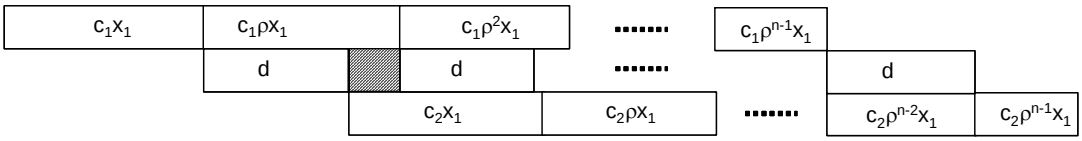
2.3 Uncapacitated ILI

When $\rho > 1$ and the item machines are dominant, we have the following optimal schedule that maximizes the unit processing time of the lot machine.



Theorem 6 (Item Machine Dominance for ILI and $\rho>1$). For the uncapacitated 3-machine MOJ problem where machines M_1 and M_3 are item-processing machines and machine M_2 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1>1$. Then the item-processing machines dominate the lot-processing machine if and only if $d \le c_2 P \frac{\rho-1}{\rho^n-1}$. Furthermore, the makespan for any such schedule is $c_1 P \frac{\rho-1}{\rho^n-1} + d + c_2 P$.

When $\rho < 1$ and item machines are dominant, we have the following optimal schedule that maximizes the unit processing time of the lot machine.



Theorem 7 (Item Machine Dominance for ILI and $\rho<1$). For the uncapacitated 3-machine MOJ problem where machines M_1 and M_3 are item-processing machines and machine M_2 is a lot-processing machine, let n be the maximum number of jobs in which to distribute P units of required processing. Let $\rho=c_2/c_1<1$. Then the item-processing machines dominate the lot-processing machine if and only if $d \le c_2 \rho^{n-2} P \frac{\rho-1}{\rho^n-1}$. Furthermore, the makespan for any such schedule is $c_1 P \frac{\rho-1}{\rho^n-1} + d + c_2 P$.

2.4 Three Machine Summary

All configurations for three machines with two item machines have been addressed except for ρ=1. When the speeds of the item machines are equal, then the optimal sizes of all jobs are equal. Regardless of the position of the lot machine, the item machines are dominant when the unit processing time of the lot machine does not exceed the duration of any job executing on an item machine, i.e., c_2P/F.

Theorem 8 (Item Machine Dominance $\rho=1$). When the two item machines operate at the same speed, i.e., $\rho=1$, then the optimal job size for all jobs is P/n. Hence, the item machines dominate if and only if $d \le c_2 P / n$

The following tables summarize the results for mixed-mode processing with three machines.

Table 1: Conditions for Two Item Machines Dominating a Lot Machine

	ρ < 1	ρ = 1	ρ > 1
IIL	$d \le c_2 \rho^{n-1} P \frac{\rho - 1}{\rho^n - 1}$	$d \le c_2 P / n$	$d \le \frac{P c_2 \rho \left(\rho^{n-1} - 1\right)}{(n-1)\left(\rho^n - 1\right)}$
ILI	$d \le c_2 \rho^{n-2} P \frac{\rho - 1}{\rho^n - 1}$	$d \le c_2 P / n$	$d \le c_2 P \frac{\rho - 1}{\rho^n - 1}$
LII	$d \le \frac{P c_1 \left(\rho^{n-1} - 1\right)}{(n-1)\left(\rho^n - 1\right)}$	$d \le c_2 P / n$	$d \le c_1 P \frac{\rho - 1}{\rho^n - 1}$

Corollary 2. The optimal makespan for any item-machine dominating 3-machine flowshop with one lot machine is $c_1 P \frac{\rho - 1}{\rho^n - 1} + c_2 P + d$.

Figure 3 plots the maximum unit processing time for the lot machine to ensure item machine dominance as a function of ρ, by varying c_2 while holding other values constant: c_1=10, P=100, n=6.

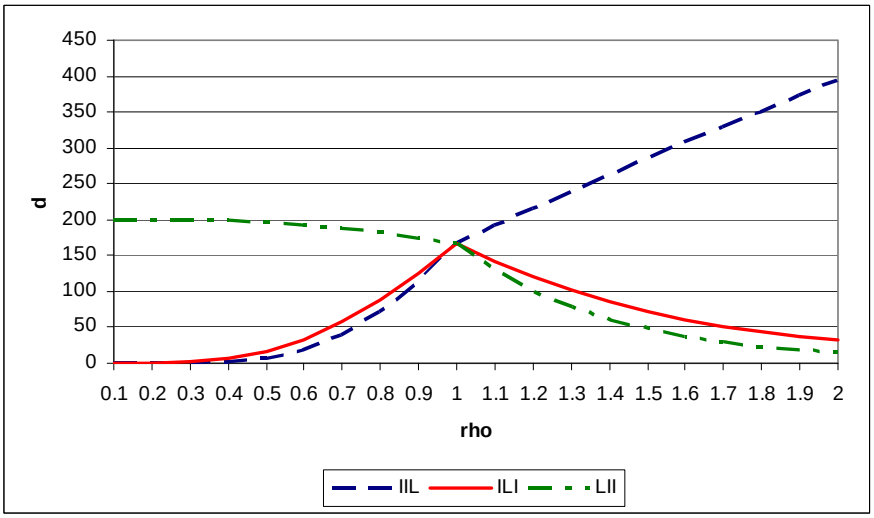


Figure 3. Maximum unit processing time for a lot machine under item-machine dominance

Given a set of two item machines and one lot machine where ρ<1, the bound d that determines whether the item machines dominate increases if the lot machine is moved back towards the beginning of the flowshop, i.e., $d_{IIL} < d_{ILI} < d_{LII}$. We claim that the makespan for any item-machine dominating configuration is at least a good as that for a configuration of the same machines that is not item-dominating. Hence, if there is a choice in placement of the lot machine, it should be placed upstream when ρ<1 (and downstream when ρ>1).

In Figure 4, the number of available jobs has been increased from 6 to 25. As the number of jobs increases, the conditions for item machine dominance become much more restrictive when ρ<1 in an IIL configuration, and when ρ>1 in an LII configuration.

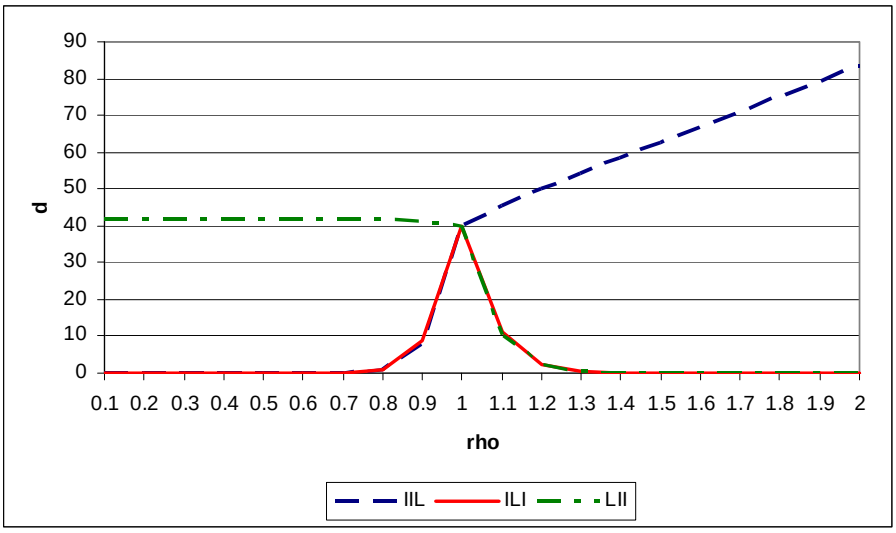


Figure 4. Item Machine Dominance Conditions for 25 Jobs

3. Conclusions

Optimizing the makespan for the multiple-orders-per-job problem requires both job formation and job scheduling, while adhering to restrictions of limited job capacity and limited number of jobs allowed, without allocating any order across multiple jobs. In a multiple-orders-per-job problem, each machine is either item-processing, where the job processing time is proportional to the total number of items in all orders within a job, or lot-processing, where the job processing time is not dependent on job size. Prior results have provided heuristics for flowshops of two and three machines, where each machine is item-processing. In this paper, we examine three-machine mixed flowshops, where there is a mix of both item-processing machines and lot-processing machines. We provide formulas for the conditions under which the item-processing machines dominate the lot-processing machine, and vice versa. Under machine domination, the optimal job sizes, assuming infinite order divisibility, are equivalent to smaller problems previously solved. This results in simple calculation of lower bounds for the optimal solution, as well as target job sizes for previously defined heuristics. We also provide a complete characterization of the relationships between the lot-processing speed and the ratio of the item-processing speeds under machine domination conditions. In future work, the authors will present heuristics and experimental results for three machine problems and discuss extensions of our approach for use in mixed flowshops with more than three machines.

References

[1] C.N. Potts and K.R. Baker (1989), Flow Shop Scheduling with Lot Streaming, *Operations Research Letters* **8**, 297 – 303.

[2] J.D. Laub, J.W. Fowler, A.B. Keha (2005), Minimizing Makespan With Multiple Orders Per Job In A Two Machine Flowshop, *European Journal of Operational Research* **182**, 63 – 79.

[3] J.D. Laub, J.W. Fowler, A.B. Keha (2006), Capacitated Lot Streaming in 3-Machine Flow Shops. ASU Working Paper ASUIE-ORPS-2006-016.

[4] J.D. Laub, J.W. Fowler, A.B. Keha (2006a). Minimizing Makespan with Multiple Orders per Job in Flowshops with Item-Processing Machines, ASU Working Paper ASUIE-ORPS-2006-017.

Estimation of Absolute Error for Minimization Maximum Lateness

Alexander A. Lazarev
Computing Centre of the Russian Academy of Sciences, Vavilov str., 40, 119991,GSP-1, Russia, alaz@ccas.ru

In this paper, we consider the approach finding of the approximate decision with the guaranteed absolute error for the problems minimizing maximum lateness. The idea of the approach consists in construction to a initial instance A such instance B (with the same number of jobs) with minimum of estimation of absolute error that

$$0 \leq L^A_{max}(\pi^B) - L^A_{max}(\pi^A) \leq \rho_d(A,B) + \rho_r(A,B) + \rho_p(A,B),$$

where

$$\rho_d(A,B) = \max_{j\in N}\{d^A_j - d^B_j\} + \max_{j\in N}\{d^B_j - d^A_j\},$$

$$\rho_r(A,B) = \max_{j\in N}\{r^A_j - r^B_j\} + \max_{j\in N}\{r^B_j - r^A_j\}$$

and

$$\rho_p(A,B) = \sum_{j\in N} |p^A_j - p^B_j|,$$

and π^A, π^B – optimal schedules for instances A and B, respectively. Besides $\rho(A,B) = \rho_d(A,B) + \rho_r(A,B) + \rho_p(A,B)$ satisfies to properties of the metrics in $(3n-2)$-dimensional space $\{(r_j, p_j, d_j) \,|\, j \in N\}$ with fixed in two parameters.

Keywords: Algorithmics, Machine Scheduling, Multi-processor Scheduling, Theoretical Scheduling, Estimation Absolute Error.

Introduction

Suppose that m identical machines $M_i, i = 1, \ldots, m$, have to process n jobs $J_j, j \in N = \{1, \ldots, n\}$. Preemption of the jobs is not allowed. The machines can handle only one job at a time. For each job j, $j \in N$, a release date r_j, a processing time $p_j > 0$ and a due date d_j are given. The precedence relations between jobs may be represented by an acyclic directed graph G.

A schedule π is uniquely determined by a permutation of the elements of N, which consists of m schedules π_i for each machine $M_i, i = 1, \ldots, m, \pi = \bigcup_{i=1}^{m} \pi_i$. The objective function is maximum lateness $L_{max}(\pi) = \max_{j\in N} L_j(\pi)$, where $L_j(\pi) = C_j(\pi) - d_j$, and $C_j(\pi)$ is complete time job $j \in N$ in schedule π.

This problem $P|prec; r_j|L_{max}$ is a generalisation of some unary NP-hard problems, for example: $P|intree; r_j; p_j = 1|C_{max}$, $P|outtree; p_j = 1|L_{max}$ [1]; $P2|chains|C_{max}$ [2]; $P||C_{max}$ [3]; $1|r_j|L_{max}$, $P2||C_{max}$ [4]; $P|prec; p_j = 1|C_{max}$ [5].

For some related problems exist polynomial solvable cases: $P2|prec; r_j; p_j = 1|L_{max}$ [6]; $P|p_j = p; r_j|L_{max}$ [7]; $1|prec; pmtn; r_j|L_{max}$ [8]; $P|chains; r_j; p_j = 1|L_{max}$ [9]; $1|prec; pmtn; r_j|L_{max}$ [10]; $P|chains; r_j; p_j = 1|L_{max}$ [11]; $P2|prec; p_j = p|L_{max}$ [12]; $JMPM|prec; r_j; n = 2|L_{max}$ [13]; $1|prec|L_{max}$ [14]; $1|prec; p_j = p; r_j|L_{max}$ [15].

Estimation of an absolute error for the problem minimizing maximum lateness for single machine $1|r_j|L_{max}$ has been considered in [16, 17].

1 Definitions

Definition 1 *We denote by $L_j^A(\pi)$ and $C_j^A(\pi)$ lateness and complete time of job j in schedule π for instance A with parameters $\{G^A, (r_j^A, p_j^A, d_j^A) | \, j \in N\}$. And, accordingly, $L_{max}^A(\pi) = \max_{j \in N} L_j^A(\pi)$ and π^A– optimal schedule for instance A.*

Definition 2 *Let's name an instance $B = \{G^B, (r_j^B, p_j^B, d_j^B) \, | \, j \in N\}$* **inverse** *to initial instance $A = \{G^A, (r_j^A, p_j^A, d_j^A) \, | \, j \in N\}$, if*

$$r_j^B = -d_j^A, \;\; p_j^B = p_j^A, \;\; d_j^B = -r_j^A, \;\; \forall \; j \in N.$$

In instance B orientation of all edges of the graph is replaced on opposite, $\overleftarrow{G}^B = \overrightarrow{G}^A$. And schedule $\pi_i' = (j_{n_i}, j_{n_i - 1}, \ldots, j_1)$ is **inverse** *to schedule $\pi_i = (j_1, \ldots, j_{n_i - 1}, j_{n_i})$ for each machine $i \in M$.*

Definition 3 *For two any instances A and B we'll define following functions:*

$$\rho_d(A, B) = \max_{j \in N} \{d_j^A - d_j^B\} + \max_{j \in N} \{d_j^B - d_j^A\},$$

$$\rho_r(A, B) = \max_{j \in N} \{r_j^A - r_j^B\} + \max_{j \in N} \{r_j^B - r_j^A\},$$

$$\rho_p(A, B) = \sum_{j \in N} |p_j^A - p_j^B|,$$

$$\rho(A, B) = \rho_d(A, B) + \rho_r(A, B) + \rho_p(A, B).$$

As for instances $A = \{G, (r_j, p_j, d_j) \, | \, j \in N\}$ and $A' = \{G, (r_j + \alpha, p_j, d_j + \beta) \, | \, j \in N\}$ the set of optimum schedules equals, it is possible "to fix" two parameters, for definiteness, $\alpha = -r_1$ and $\beta = -d_1$. Then the function $\rho(A, B)$ satisfies to properties of normed metrics in $(3n-2)$-dimensional space.

2 Estimation of absolute error

Lemma 1 *Let $A = \{G^A, (r_j, p_j, d_j^A) | \, j \in N\}$ and $B = \{G^B, (r_j, p_j, d_j^B) | \, j \in N\}$ (with identical release and processing times $r_j, p_j, j \in N$,) are two instances then for any schedule π holds*

$$L_{max}^B(\pi) - L_{max}^A(\pi) \leq \max_{j \in N} \{d_j^A - d_j^B\}. \tag{1}$$

Proof: For any $j \in N$ we have: $L_{max}^A(\pi) + \max_{i \in N} \{d_i^A - d_i^B\} \geq C_j(\pi) - d_j^A + d_j^A - d_j^B = C_j(\pi) - d_j^B$. So, $L_{max}^A(\pi) + \max_{i \in N} \{d_i^A - d_i^B\} \geq \max_{j \in N} \{C_j(\pi) - d_j^B\} = L_{max}^B(\pi)$. □

The instances A and B are "symmetric", so obviously for any schedule π holds

$$L_{max}^A(\pi) - L_{max}^B(\pi) \leq \max_{j \in N} \{d_j^B - d_j^A\}. \tag{2}$$

Lemma 2 *Let $A = \{G, (r_j, p_j, d_j^A) | \, j \in N\}$ and $B = \{G, (r_j, p_j, d_j^B) | \, j \in N\}$ (with identical release and processing times $r_j, p_j, j \in N$, and graph G) are two any instances then*

$$0 \leq L_{max}^A(\pi^B) - L_{max}^A(\pi^A) \leq \rho_d(A, B).$$

Proof: From (1), (2) for schedules π^A and π^B we have

$$L^A_{max}(\pi^A) + \max_{j\in N}\{d^A_j - d^B_j\} \geq L^B_{max}(\pi^A), \tag{3}$$

$$L^B_{max}(\pi^B) + \max_{j\in N}\{d^B_j - d^A_j\} \geq L^A_{max}(\pi^B). \tag{4}$$

Schedule π^B is optimal for instance B so

$$L^B_{max}(\pi^A) \geq L^B_{max}(\pi^B). \tag{5}$$

¿From (3)–(5)

$$L^A_{max}(\pi^A) + \max_{j\in N}\{d^A_j - d^B_j\} \geq L^A_{max}(\pi^B) - \max_{j\in N}\{d^B_j - d^A_j\},$$

then

$$L^A_{max}(\pi^A) + \rho_d(A,B) \geq L^A_{max}(\pi^B) \geq L^A_{max}(\pi^A).$$

□

The instances A and B are "symmetric" so obviously $L^B_{max}(\pi^A) - L^B_{max}(\pi^B) \leq \rho_d(A,B) = \rho_d(B,A)$.

Lemma 3 *Let A and B are inverse instances and π and π' – inverse schedules, then $L^A_{\max}(\pi^A) = L^B_{\max}(\pi^B)$.*

Lemma 4 *Let $A = \{G^A, (r^A_j, p_j, d^A_j) | \, j \in N\}$ and $B = \{G^B, (r^B_j, p_j, d^B_j) | \, j \in N\}$ (with identical processing times $p_j, j \in N$,) are two any instances then*

$$0 \leq L^A_{max}(\pi^B) - L^A_{max}(\pi^A) \leq \rho_r(A,B). \tag{6}$$

Lemma 5 *Let $A = \{G, (r_j, p^A_j, d_j) | \, j \in N\}$ and $B = \{G, (r_j, p^B_j, d_j) | \, j \in N\}$ (with identical release times and due dates $r_j, d_j, j \in N$, and graph preceding G) are two any instances then*

$$0 \leq L^A_{max}(\pi^B) - L^A_{max}(\pi^A) \leq \sum_{j\in N} |p^A_j - p^B_j| = \rho_r(A,B). \tag{7}$$

Theorem 1 *Let $A = \{G, (r^A_j, p^A_j, d^A_j) | \, j \in N\}$ and $B = \{G, (r^B_j, p^B_j, d^B_j) | \, j \in N\}$ (with identical graph preceding G) are two any instances then*

$$0 \leq L^A_{max}(\pi^B) - L^A_{max}(\pi^A) \leq \rho(A,B). \tag{8}$$

¿From "symmetric" the instances A and B holds

$$0 \leq L^B_{max}(\pi^A) - L^B_{max}(\pi^B) \leq \rho(A,B) = \rho(B,A). \tag{9}$$

Theorem 2 *Let $A = \{G, (r^A_j, p^A_j, d^A_j) | \, j \in N\}$ and $B = \{G, (r^B_j, p^B_j, d^B_j) | \, j \in N\}$ (with identical graph preceding G) are two any instances then*

$$0 \leq L^A_{max}(\overline{\pi}) - L^A_{max}(\pi^A) \leq \delta^B(\overline{\pi}) + \rho(A,B), \tag{10}$$

where $\delta^B(\overline{\pi}) = L^B_{max}(\overline{\pi}) - L^B_{max}(\pi^B)$.

3 The scheme of approached decision of the problem

The idea of the approached decision of the problem consists of two stages. On the first step to the initial instance $A = \{G, (r_j^A, p_j^A, d_j^A) | \, j \in N\}$ is such change of its parameters r_j, p_j and d_j that the received instance $B = \{G, (r_j^B, p_j^B, d_j^B) | \, j \in N\}$ is belonged to a set polynomial solvable instances of the initial problem. On the next step we'll find optimal schedule to instance B. According to theorem 1 the schedule π^B to instance A have $0 \le L_{max}^A(\pi^B) - L_{max}^A(\pi^A) \le \rho(A, B)$.

Let's consider a case when class of polynomial solvable instances of the problem is defined by system of k linear inequalities

$$\mathbf{X} * \mathbf{R} + \mathbf{Y} * \mathbf{P} + \mathbf{Z} * \mathbf{D} \le \mathbf{H}, \tag{11}$$

(s. t. $p_j \ge 0$, $\forall j \in N$), where $R = (r_1, \ldots, r_n)^T$, $P = (p_1, \ldots, p_n)^T$, $D = (d_1^C, \ldots, d_n)^T$, and $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ – matrixes of dimension $k \times n$, and $H = (h_1, \ldots, h_k)^T$ – k-dimensional vector (the up index ^{T} designates transposition). Then in this class of instances (11) we find instance B with minimum "distance" $\rho(A, B)$ (to the initial instance A),

$$\begin{cases} \min\,(x^d - y^d + x^r - y^r) + \sum_{j \in N} x_j^p \\ y^d \le d_j^A - d_j^B \le x^d, \quad \forall j \in N, \\ y^r \le r_j^A - r_j^B \le x^r, \quad \forall j \in N, \\ -x_j^p \le p_j^A - p_j^B \le x_j^p, \quad \forall j \in N, \\ 0 \le x_j^p, \quad \forall j \in N, \\ \mathbf{X} * \mathbf{R^B} + \mathbf{Y} * \mathbf{P^B} + \mathbf{Z} * \mathbf{D^B} \le \mathbf{H}. \end{cases} \tag{12}$$

Linear programming problem (12) with $3n + 4 + n$ variables ($r_j^B, p_j^B, d_j^B, j = 1, \ldots, n$, and x^d, y^d, x^r, y^r, and x_j^p, $j = 1, \ldots, n$) and $7n + k$ inequalities can be sometimes solved in polynomial time considering specificity of linear restrictions. For example, for the problem $1|r_j|L_{max}$ two cases have been allocated polynomial solvable instances:

$$\begin{cases} d_1 \le \ldots \le d_n, \\ d_1 - r_1 - p_1 \ge \ldots \ge d_n - r_n - p_n \end{cases} \tag{13}$$

and

$$\max_{k \in N} \{d_k - r_k - p_k\} \le d_j - r_j, \, \forall j \in N. \tag{14}$$

In the case (13) of the problem $1|r_j|L_{max}$ can be solved for polynomial time – $O(n^3 \log n)$ operations [16], [18]. And the task of linear programming (12) has been can be solved for polynomial time – $O(n \log n)$ operations [16], [17]. In the case (14) the problem can be solved in $O(n^2 \log n)$ operations [19]. As well as in case of (13) the minimum of absolute error of maximum lateness can be solved for polynomial time – in $O(n)$ operations [17].

If does not exist polynomial solvable instances or the "distance" $\rho(A, B)$ to any polynomial solvable point B is too "great", but we know for some instance $B = \{G, (r_j^B, p_j^B, d_j^B) | \, j \in N\}$ the absolute error of maximum lateness, then for initial instance $A = \{G, (r_j^A, p_j^A, d_j^A) | \, j \in N\}$ we can find approximation schedule $\overline{\pi}$ with the guaranteed absolute error of maximum lateness $0 \le L_{max}^A(\overline{\pi}) - L_{max}^A(\pi^A) \le \delta^B(\overline{\pi}) + \rho(A, B)$, according to theorem 2.

4 The scheme of approached decision for "close" problems

Let's need to find optimal schedule for some instance A to the problem $\alpha|\beta|L_{\max}$ and we know that the corresponding problem $\alpha|\beta|C_{\max}$ is polynomially solvable. Then all parameters except d_j

$(d_j^B = 0), \forall j \in N$, of instance B will be the analogous. Thus

$$0 \le L_{\max}^A(\pi^B) - L_{\max}^A(\pi^A) \le \rho(A,B) = \max_{j\in N} d_j^A - \min_{j\in N} d_j^A.$$

Then let's need to find optimal schedule for instance A to the problem $\alpha|\beta|L_{\max}$ and we know that the corresponding problem $\alpha|\beta, p_j = p|L_{\max}$ is polynomially solvable. All parameters except $p_j, \forall j \in N$, of instance B will be the same. Thus we should decide optimization task

$$\rho(A,B) = \sum_j |p_j - p| \to \min_p.$$

Solution of the task is $p^* = p_{[\frac{n+1}{2}]}$ (if $p_1 \le \ldots \le p_n$). So we draw

$$L_{\max}^A(\pi^B) - L_{\max}^A(\pi^A) \le \sum_{j=1}^{n} \left| p_j - p_{[\frac{n+1}{2}]} \right|.$$

If there is constrain $p_j = 1$, instead of $p_j = p, \forall j \in N$, then absolute error of maximum lateness to meet next condition

$$L_{\max}^A(\pi^B) - L_{\max}^A(\pi^A) \le \sum_{j=1}^{n} \left| p_j - p_{[\frac{n+1}{2}]} \right| + 2p_{[\frac{n+1}{2}]}.$$

Let's our problem is $R|\beta|L_{\max}$ or $Q|\beta|L_{\max}$ then for the initial instance A of the problem we consider the problem $P|\beta|L_{\max}$ that is polynomially solvable. All parameters except $p_{ji}, \forall j \in N, \forall i \in M$, of instance B will be the same. Thus we should decide next optimization task

$$\rho(A,B) = \sum_{j\in N} \left(\max_{i\in M}\{(p_{ji}^A - p_j^B), 0\} - \min_{i\in M}\{(p_{ji}^A - p_j^B), 0\} \right) \to \min_{p_j}.$$

So we can take any value for p_j^B from interval $[\min_{i\in M} p_{ji}^A, \max_{i\in M} p_{ji}^A], \forall j \in N$, and

$$L_{\max}^A(\pi^B) - L_{\max}^A(\pi^A) \le \sum_{j\in N} \left(\max_{i\in M} p_{ji}^A - \min_{i\in M} p_{ji}^A \right).$$

Let's need to find optimal schedule for instance A to the $R|\beta|L_{\max}$ and we know that the corresponding problem $Q|\beta|L_{\max}$ is polynomially solvable. Then all parameters except $p_{ji}, \forall j \in N, \forall i \in M$, of instance B will be the same. In Q-problem $p_{ji} = p_j\sigma_i, \forall j \in N, \forall i \in M$. Thus we need to decide next optimization task

$$\rho(A,B) = \sum_{j\in N} \left(\max_{i\in M}\{(p_{ji}^A - p_j^B\sigma_i^B), 0\} - \min_{i\in M}\{(p_{ji}^A - p_j^B\sigma_i^B), 0\} \right) \to \min_{p_j^B, \sigma_i^B}.$$

The task can be represented as linear programming problem and be decided by simplex-method:

$$\begin{cases} \sum_{j\in N} (\alpha_j - \beta_j) \to \min\limits_{\alpha_j, \beta_j, p_j^B, \sigma_i^B} \\ \beta_j \le p_{ji}^A - p_j^B\sigma_i^B \le \alpha_j, \qquad \forall j \in N, \forall i \in M, \\ \beta_j \le 0 \le \alpha_j, \qquad \forall j \in N. \end{cases}$$

The question of change the graph preceding is not considered yet...

References

[1] P. Brucker, M.R. Garey, D.S. Johnson (1977), Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness, *Math. Oper. Res.* **2**(3), 275 – 284.

[2] J. Du, J.Y.-T. Leung, G.H. Young (1991), Scheduling chain-structured tasks to minimize makespan and mean flow time, *Inform. and Comput.* **92**(2), 219 – 236.

[3] M.R. Garey, D.S. Johnson (1978), "Strong" NP-completeness results: motivation, examples, and implications, *J. Assoc. Comput. Mach.* **25**(3), 499 – 508.

[4] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker (1977), Complexity of machine scheduling problems, Studies in integer programming (Proc. Workshop, Bonn, 1975), *Ann. of Discrete Math.* **1**, 343 – 362.

[5] J.D. Ullman (1975), NP-complete scheduling problems, *J. Comput. System Sci.* **10**, 384 – 393.

[6] M.R. Garey, D.S. Johnson (1977), Two-processor scheduling with start-times and deadlines, *SIAM J. Comput.*, **6**(3), 416 – 426.

[7] B. Simons (1983), Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines, *SIAM J. Comput.*, **12**(2), 294 – 299.

[8] J. Blazewicz (1976), Scheduling dependent tasks with different arrival times to meet deadlines, *Model. Perform. Eval. Comput. Syst., Proc. int. Workshop, Stresa*, 57 – 65.

[9] P. Baptiste, P. Brucker, S. Knust, V. Timkovsky (2004), Ten notes on equal-execution-time scheduling, *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* **2**, 111 – 127.

[10] K.R. Baker, E.L. Lawler, J.K. Lenstra, A.H.G Rinnooy Kan (1983), Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints, *Oper. Res.* **31**, 381 – 386.

[11] M. Dror, W. Kubiak, P. Dell' Olmo (1998), Strong-weak chain constrained scheduling, *Ricerca Operativa* **27**, 35 – 49.

[12] M.R. Garey, D.S. Johnson (1976), Scheduling tasks with nonuniform deadlines on two processors, *J. Assoc. Comput. Mach.* **23**, 461 – 467.

[13] B. Jurisch (1995), Lower bounds for the job-shop scheduling problem on multi-purpose machines, *Discrete Appl. Math., Discrete Applied Mathematics. Combinatorial Algorithms, Optimization and Computer Science* **58**(2), 145 – 156.

[14] E.L. Lawler (1973), Optimal sequencing of a single machine subject to precedence constraints, *Management Sci.* **19**, 544 – 546.

[15] B. Simons (1978), A fast algorithm for single processor scheduling, *19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich.)*, 246 – 252.

[16] A.A. Lazarev (1989), Efficient algorithms of decisions some problems of scheduling theory for single machine with due dates of service jobs, *PhD Dissertation*: 108 p (in Russian).

[17] A.A. Lazarev, R.R. Sadykov, S.V. Sevastianov (2006), The scheme of the approached decision of a single machine to minimize maximum lateness, *The discrete analysis and operations research* **2, 13, 1**: 57 – 76 (in Russian).

[18] A.A. Lazarev (2006), The Pareto-Optimal Set of the NP-Hard Problem of Minimizing of the Maximum Lateness for a Single Machine, *Journal of Computer and System Sciences International* **45**(6), 943 – 949.

[19] J.A. Hoogeveen (1996), Minimizing maximum promptness and maximum lateness on a single machine, *J. Math. Oper. Res.* **21**(1), 100 – 114.

Scheduling the Operations of an Integrated Production-Distribution Process

Lei Lei
Rutgers Business School, Rutgers University, Newark, NJ, USA, llei@andromeda.rutgers.edu
Wanpracha Art Chaovalitwongse
Department of Industrial and Systems Engineering, Rutgers University, Piscataway, NJ, USA, wchaoval@rci.rutgers.edu
Selim Bora
Center for Operations Research, Rutgers University, Piscataway, NJ, USA, sbora@eden.rutgers.edu

We consider a variation of the integrated scheduling problem where the operations of a capacitated facility and a fleet of heterogeneous vessels must be sequenced to minimize the operating cost during a multi-period production and distribution process. We prove the NP-hardness of the problem, analyze the problem properties, and propose a partial linear-programming relaxation based heuristic approach. We also derive an error bound of this heuristic for a special case of the problem. Finally, the empirical observations on the computational performance of the heuristic under general cases are reported.

Keywords: Integrated production and distribution, heterogeneous vessels, berthing and discharging time, multi-level capacities, heuristic, error bound, empirical performances.

1. Introduction

The cost to produce and deliver gasoline products to the market consists of three major components: the cost of crude oil to refiners, the cost of refinery processing, and the cost of marketing and distribution [13]. As the U.S. retail gasoline prices continue to rapidly elevate, effectively coordinating the demand and supply of gasoline products has become ever more crucial to large oil companies [6].

Our study is focused on the demand-supply coordination problem motivated by operational issues faced by a major U.S. corporation in petrochemical industry. The company operates a number of capacitated oil refineries and uses its own and chartered vessels to distribute the gasoline products to discharging/demand locations along the coastline of North America. Each refinery is capable of producing a number of products for its commercial and industry users and carries inventories for those produced but not yet being distributed. These inventories also serve as reservoirs to supply the demands during peak seasons. However, none of the inventories can exceed its maximum holding capacity as which would halt the production. Each discharging location carries its own inventories and serves as a depot of distribution for the local market. Since vessels are expensive in both variable and fixed costs, any inefficiency in the supply process could result in a substantial operating cost. The integrated production and distribution scheduling problem encountered in this process is very complicated due to the involvement of heterogeneous vessels (*e.g.,* in terms of their loading capacities, discharging and berthing times, and operating costs) and the fact that each vessel has multi-level of loading capacities such that a load beyond the normal/base capacity will result in an extra overload cost. Practical issues faced by the company include how much should be produced in each time period (typically 2 weeks), which vessel should deliver to which depot in which time period, whether a particular vessel trip should carry an extra load and by how much, and what should be the ending inventory at a depot in a particular period, *etc*. Due to high distribution cost of gasoline products, an effectively integrated production and distribution schedule could help the company to further improve the profit of its supply chain and to strengthen its competitive advantage in the market place.

Optimally solving such an integrated operations scheduling problem is not an easy task. It differs from the classical lot-sizing [2,8] and/or inventory-routing problems [4-5,7,9-11,14] due to the heterogeneity of vessels, multi-level of loading capacities, finite production rate, and non-negligible berthing and discharging time at the customer ports. In the gasoline distribution between local refineries and customer ports, the berthing and discharging times are usually very significant comparing to the vessel traveling time due to the overhead time of connecting/disconnecting the oil pipes at ports. These inventory management and distribution processes are often referred to an integrated inventory scheduling problem, which is quite different from a traditional inventory routing problem [1,3].

We consider in this study a variation of this integrated inventory scheduling problem. In particular, we focus on the case involving a single capacitated refinery with a maximum inventory capacity $I_0^{\max}$ (thousand tons), and a maximum production rate $\pi^{\max}$ (thousand tons per period). There is a fleet of heterogeneous vessels, $v \in V$, that are used transport products from the refinery to the discharging locations, or customer ports, $n \in N$, in a given distribution network. Each customer port, *n*, has a pre-specified order/demand $d_{n,t}$ (thousand tons) needed to serve its local demand in period *t*, $1 \le t \le T$. Each vessel *v* has two levels of capacities: the normal/base loading capacity u_v^0 (thousand tons) and the maximum absolute capacity $u_v^{\max}$, so that carrying a load more than $u_v^{\max}$ is infeasible and carrying a load more than u_v^0 results in an additional operating (overload) cost c_v^o dollars per one thousand tons. Each individual vessel has a non-negligible berthing times, $b_{v,n} > 0$, at port *n* and a non-negligible unloading time determined by the discharging rate r_n (thousand tons per day) at port *n, n=1,2,…,N*. In addition, we have the following assumptions, which are based on standard practice of oil logistics operations, that $d_{n,t} \le u_v^{\max}$, $\forall v, \forall n, \forall t$, and backlogging is allowed but penalized at p_n (thousand dollars per thousand tons, per period), *n=1,2,…,N*. The orders cannot be split; that is, the demand quantity $d_{n,t}$ cannot be shared by more than one vessel. The total berthing and discharging time of a vessel cannot exceed a given portion of operating time, β, where $0 < \beta < 100\%$, per time period. Let τ denote the length of each time period (days). We also assume that the vessel traveling operations, due to the significance of vessel berthing and discharging times, are simplified (and thus not explicitly modeled in the analysis) by reserving a portion of time, $(1-\beta)\cdot\tau$, per time period for the vessel traveling. Furthermore, there is a fixed cost c_v^f (in thousand dollars) for utilizing any vessel *v* in a time period, and there are *T* time periods in the given planning horizon. The objective of this problem is to determine an integrated operation schedule for production, inventory, and distribution so that the sum of vessel fixed cost, overload cost, inventory holding cost, and product shortage cost is minimized, subject to the vessel, production, and inventory capacity constraints. We shall use ***P*** to denote this integrated scheduling problem.

In this paper, we analyze the properties of ***P*** and then propose a partial linear programming (LP) relaxation based heuristic that solves ***P*** through an iterative process. At each iteration, the heuristic relaxes the integer requirement (integrality constraint) on a subset of integer variables and optimally solves the partially relaxed problem. The integer portion of the solutions is retained while the remaining part of the problem is handled in the following iteration. The iteration continues until a complete set of integer solutions are obtained. In Section 2, we give a formal definition of Problem ***P***, prove its complexity, and introduce a special case with *T=1* that is strongly polynomial time solvable. We then show that the error gap between the partial LP relaxation based heuristic solution and the optimal solution is bounded under certain conditions. In Section 3, we provide a thorough analysis on the computational performance of this proposed heuristic under five

test scenarios, including the case with several real life problem parameters. Finally in Section 4, we conclude the study and discuss future extensions.

2. Properties of Problem *P*

To formally define Problem ***P*** for any given planning horizon of *T* periods with *N* customer depots and a fleet of *V* heterogeneous vessels, let $v = 1,\ldots,V$, $n = 1,\ldots,N$, $t = 1,\ldots,T$, and

$Y_{v,n,t}$ Binary variables, and $Y_{v,n,t}=1$ if vessel *v* visits port *n* in period t;
$Q_{v,n,t}$ Quantity carried by vessel *v* in period *t* for port *n*;
$Z_{v,t}$ Binary variables, and $Z_{v,t}=1$ if vessel *v* is utilized in period t;
$I_{n,t}$ Ending inventory of port *n* in period *t*;
$S_{n,t}$ Amount of shortage at port *n* in period *t;*
$I_{0,t}$ Ending inventory of the refinery in period *t;*
P_t Quantity being produced by the refinery in period *t;*

[Model *P*]

$$\textit{Minimize} \quad \sum_{\forall v} c_v^f \cdot \sum_{\forall t} Z_{v,t} + \cdot \sum_{\forall v}\sum_{\forall t} c_v^o \cdot \max\left\{0, \sum_{\forall n} Q_{v,n,t} - u_v^0\right\} + h_0 \cdot \sum_{\forall t} I_{0,t} + \sum_{\forall n} h_n \cdot \sum_{\forall t} I_{n,t} + \sum_{\forall n} p_n \cdot \sum_{\forall t} S_{n,t} \tag{1}$$

s.t.

$$\sum_{\forall n}\left(b_{v,n} \cdot Y_{v,n,t}\right) + 2 \cdot \sum_{\forall n} Q_{v,n,t} / r_n \le \beta \cdot \tau \qquad \forall v = 1,\ldots,V, \forall t = 1,\ldots,T \tag{2}$$

$$\sum_{\forall n} Q_{v,n,t} \le u_v^{\max} \qquad \forall v = 1,\ldots,V, \forall t = 1,\ldots,T \tag{3}$$

$$I_{n,t} = I_{n,t-1} + \sum_{\forall v} Q_{v,n,t} - d_{n,t} + S_{n,t} \qquad \forall n = 1,\ldots,N, \forall t = 1,\ldots,T \tag{4}$$

$$I_{0,t} = I_{0,t-1} + P_t - \sum_{\forall v}\sum_{\forall n} Q_{v,n,t} \qquad \forall t = 1,\ldots,T \tag{5}$$

$$P_t \le \pi^{\max}, \quad I_{0,t} \le I_0^{\max} \qquad \forall t = 1,\ldots,T \tag{6a}$$

$$I_{n,t} \le I_n^{\max} \qquad \forall n = 1,\ldots,N, \forall t = 1,\ldots,T \tag{6b}$$

$$\sum_{\forall n} Y_{v,n,t} \le M \cdot Z_{v,t} \qquad \forall v = 1,\ldots,V, \forall t = 1,\ldots,T \tag{7}$$

$$Q_{v,n,t} \le M \cdot Y_{v,n,t} \qquad \forall v = 1,\ldots,V, \forall t = 1,\ldots,T, \forall n = 1,\ldots,N \tag{8}$$

$$\sum_{\forall v} Y_{v,n,t} \le 1 \qquad \forall n = 1,\ldots,N, \forall t = 1,\ldots,T \tag{9}$$

$$Y_{v,n,t}, Z_{v,t} \textit{ binary, } Q_{v,n,t}, P_t, I_{0,t}, I_{n,t} \ge 0, \qquad \forall v = 1,\ldots,V, \forall t = 1,\ldots,T, \forall n = 1,\ldots,N.$$

In Problem ***P***, the objective function in Eq. (1) is to minimize the total operation cost including (i) fixed vessel cost, (ii) overload vessel cost, (iii) inventory cost incurred at the source port, (iv) inventory costs incurred at the destination ports, and (v) shortage costs incurred at the destination ports. The constraints in Eq. (2) ensure that the total berthing and discharging time of a vessel in each time period does not exceed the total time available for the non-travel operations. The capacity constraints in Eq. (3) require that every vessel load cannot exceed the maximum loading capacity. The flow constraints in Eqs. (4)-(5) are the inventory flow/mass balance constraints for the source and the destination ports, respectively. The production capacity constraints in Eq. (6) ensure

the production and inventory quantities do not exceed the respective refinery and resource capacity. The logical constraints in Eqs. (7)-(8) ensure that a vessel is not dispatched if it is not utilized at all, and that if it is not dispatched, then it carries nothing. Finally, the constraints in Eq. (9) enforce that no more than one vessel visits each port in a given time period.

Lemma 1. *Problem **P** is NP-Hard in strong sense even with T=1 and homogeneous vessels*.

Proof. Let $\pi^{\max} = I_n^{\max} = I_0^{\max} = \infty,\ r_n = \infty,\ c_v^o = 0,\ p_n = \infty,\ T=1,\ u_v^0 = u_v^{\max},\ \forall v$. Then, this special case of Problem ***P*** is a two-dimensional bin-packing problem in which each bin has two capacities: $\sum_{\forall n}\left(b_{v,n}\cdot Y_{v,n}\right) \le \beta\cdot\tau,$ and $\sum_{\forall n} Q_{v,n} \le u_v^{\max}$, respectively. Since the one-dimensional bin packing problem is already NP-hard in strong sense, the claim holds. □

The computational effort required to solve the general version of ***P*** is excessive due to its combinatorial nature. However, under certain conditions, some of which actually hold in practices, problem ***P*** can be solved rather quickly (see [12]). One of such cases occurs when the demands at destination ports are large or when small barges, that may typically travel only for a shorter distance and serve only 1-2 ports, are used in the case of vessel shortage. Lemma 2 below proves that if *T=1* and each vessel can serve at most two destination ports, then the reduced problem can be solved in strongly polynomial time.

Lemma 2. *Problem **P** is polynomial time solvable if T=1 with homogeneous vessels and*

$$(b_{v,i} + b_{v,j} + b_{v,k}) + 2\{\frac{d_i}{r_i} + \frac{d_j}{r_j} + \frac{d_k}{r_k}\} \ge \beta\cdot\tau \qquad \forall i,j,k \in N. \qquad (10)$$

Proof. The condition in Eq. (10) implies that each vessel can serve at most two destination ports in a time period. In this case, Problem ***P*** reduces to the one of assigning no more than two ports to each vessel such that Eq. (1) is minimized[a]. The reduced problem can be transformed to the following minimum cost flow problem (see Figure 1). First, we construct a dummy source node *s* with an outgoing flow *N*, and a dummy sink node *t* with an incoming flow *N*. Denote each destination port as node *i*, for *i=1,2,…,N*, which is connected with the source node by arc *(s, i)* with a vessel fixed cost of c_v^f. Let the upper bound of the arc capacity be 2, for all arcs (*s,i*), *i=1,2,…,N*. Add a set of *N* dummy nodes *i', i'=1,2,…,N*, and arcs *(i, i')* with a cost of $C_{i,i'}$ connecting each port node *i* to every dummy node *i'*, where $i' \ge i$. Let the lower bound and upper bound of the capacity of arc *(i, i')* be 0 and 1, respectively. Connect every dummy node to the sink node *t* via arc *(i', t)* with a cost of 0. Let the lower and upper bounds of the arc capacity be both equal to 1, for all arcs *(i', t), i'=1,2,…,N*. We refer $C_{i,i'}$ as an operating cost (overload cost + shortage) of using the same vessel *to* carry the supplies for ports *i* and *i'*, which is defined by

$$C_{i,i'} = \begin{cases} a_{i,i'} & \text{if } i = i' \\ g_{i,i'} & \text{if } i < i' \end{cases}$$

where

$$a_{i,i'} = \begin{cases} c^0\cdot\min\{u^{\max}-u^0,\ \max\{0, d_i - u^0\}\} + p\cdot\max\{0,\ d_i - u^{\max}\} & \text{if } c^0 < p \\ p\cdot\max\{0,\ d_i - u^0\} & \text{if } c^0 \ge p, \end{cases} \qquad (11)$$

$$g_{i,i'} = \begin{cases} c^0\cdot\min\{u^{\max}-u^0,\ \max\{0, d_{i'} - \max\{0, u^0 - d_i\}\}\} \\ \qquad + p\cdot\max\{0,\ d_{i'} - \max\{0, u^{\max} - d_i\}\} & \text{if } c^0 < p \\ p\cdot\max\{0,\ d_{i'} - \max\{0, u^0 - d_i\}\} & \text{if } c^0 \ge p. \end{cases} \qquad (12)$$

[a] Note that the holding cost is excluded from the consideration when *T=1*.

Since the reduced minimum cost flow problem (see Figure 1) is polynomial time solvable, this finishes the proof. □

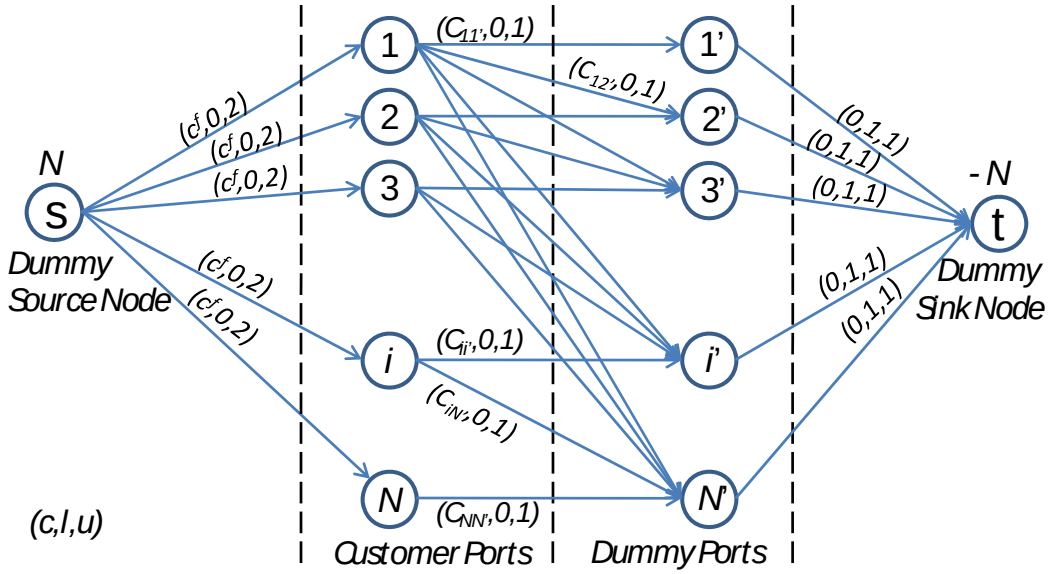


Figure 1. Transformation of Problem *P* to a minimum cost flow model.

Note that the values of $a_{i,i'}$ and $g_{i,i'}, \forall i,i'$, can be computed in a similar way even if the shortage costs are different at port *i* and port *i'* [12]. Furthermore, we have the following result.

Corollary 1. *Lemma 2 holds even in the case where we have a pool of heterogeneous vessels.*

Let *K* be the total number of vessel types. We can prove the correctness of this corollary by transforming Problem ***P*** again to a minimum cost flow problem with additional $K \times N$ *vessel nodes* (see Figure 2 for an example when *K=3* and *N=3*), where *vessel node* v_k^i, $i = 1,2,...,N$, $k = 1,2,...,K$, designates a vessel of type *k*, $1 \le k \le K$, that can be assigned to destination port *i*, $1 \le i \le N$. In addition, we define the arc cost

$$a_{i,i'}^v = \begin{cases} c_v^0 \cdot \min\{u_v^{\max} - u_v^0,\ \max\{0, d_i - u_v^0\}\} + p \cdot \max\{0,\ d_i - u_v^{\max}\} & \text{if } c_v^0 < p \\ p \cdot \max\{0,\ d_i - u_v^0\} & \text{if } c_v^0 \ge p, \end{cases} \tag{11a}$$

$$g_{i,i'}^v = \begin{cases} c_v^0 \cdot \min\{u_v^{\max} - u^0,\ \max\{0, d_{i'} - \max\{0, u_v^0 - d_i\}\}\} \\ \qquad + p \cdot \max\{0,\ d_{i'} - \max\{0, u_v^{\max} - d_i\}\} & \text{if } c_v^0 < p \\ p \cdot \max\{0,\ d_{i'} - \max\{0, u_v^0 - d_i\}\} & \text{if } c_v^0 \ge p, \end{cases} \tag{12a}$$

for the arc between vessel node v_k^i and dummy node *i'*, *i'=1,2,…,N*. The optimal solution to this extended minimal cost flow problem solves ***P*** with *T=1* and heterogeneous vessels.

We now consider another special case of Problem ***P*** with *T=2* and homogeneous vessels. We assume that $u^{\max} = u^0 = u$, $\pi^{\max} = I_0^{\max} = r_n = p_n = \infty$, $h_n < c^f / u$, $\forall n$, $h_0 = 0$, and one vessel can visit only one port in a time period. This reduces Problem ***P*** to the following problem $\boldsymbol{P_2}$.

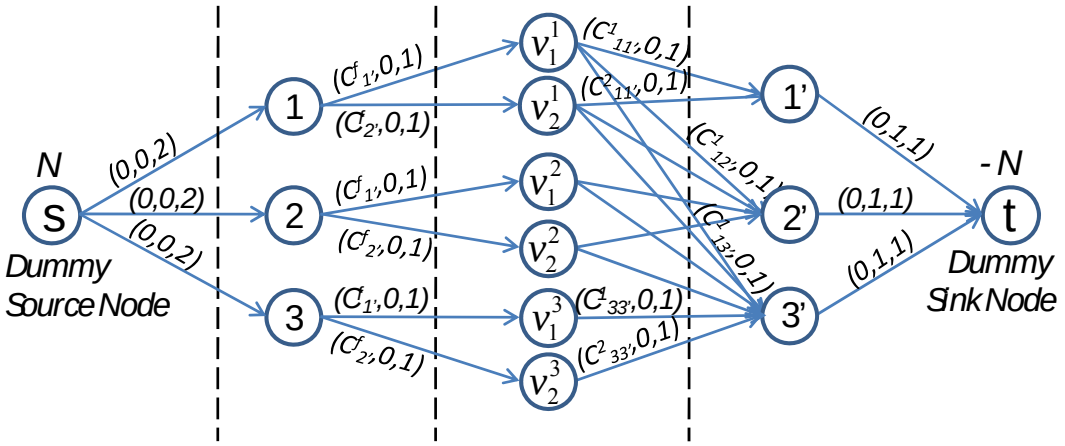


Figure 2. An example of Problem *P* minimum cost flow network with two vessel types (*K=2*) and three ports (*N=3*).

[Model P_2]

$$\text{Minimize.} \quad G(P_2) = \sum_{\forall n} h_n \cdot I_n + c^f \cdot \sum_{\forall n} \sum_{t=1,2} Z_{n,t}$$

$$\text{s.t} \quad Q_{n,1} \le u \cdot Z_{n,1} \qquad \forall n$$

$$Q_{n,2} \le u \cdot Z_{n,2} \qquad \forall n$$

$$Q_{n,1} + Q_{n,2} \ge d_{n,1} + d_{n,2} \qquad \forall n$$

$$Q_{n,1} - I_n = d_{n,1} \quad \forall n$$

$$I_n \le I_n^{\max} \qquad \forall n$$

$$Z_{n,t} \quad \text{integer}, \; Q_{n,t} \ge 0, \; I_n \ge 0, \; \forall n, \forall t$$

For any given instance of Problem $\boldsymbol{P_2}$, let us relax the integer requirements on $Z_{n,2}$, $\forall n$, and retain the integer requirement on $Z_{n,1}$, $\forall n$. Let $\boldsymbol{R_2}$ denote this partial relaxation and let $Z_{n,1}^0$, $Z_{n,2}^L$, $Q_{n,1}^L$, $Q_{n,2}^L$, I_n^L be the optimal solution to $\boldsymbol{R_2}$. Apparently, $Z_{n,1}^0$, $\lceil Z_{n,2}^L \rceil$, $Q_{n,1}^L$, $Q_{n,2}^L$, I_n^L define a feasible heuristic solution to Problem $\boldsymbol{P_2}$. Let $G^*(P_2)$, and $G^H(P_2)$, be the optimal objective function value of Problem $\boldsymbol{P_2}$, and that of the heuristic solution, respectively. Then, the gap between two objective functions $|G^*(P_2) - G^H(P_2)|$ is bounded as follows.

Lemma 3. $|G^*(P_2) - G^H(P_2)| < \sum_{\forall n} h_n \cdot \min.\{d_{n,2}, I_n^{\max}\}$.

Proof. Since $d_{n,t} \le u, \forall n, \forall t,$ and one port can be visited by at most one vessel in each time period, we have the following two possible cases for each destination port *n*: $d_{n,1} + d_{n,2} \le u$ and $d_{n,1} \le u < d_{n,1} + d_{n,2}$.

Case 1. $d_{n,1} + d_{n,2} \le u$

In this case, the optimal solution to Problem $\boldsymbol{P_2}$ is given by $Z_{n,1}^* = 1$ and

$Z_{n,2}^* = 0, \; I_n^* = d_{n,2}$ If $I_n^{\max} \ge d_{n,2}$

$Z_{n,2}^* = 1, \; I_n^* = 0$ If $I_n^{\max} < d_{n,2}$

and the optimal solution to $\boldsymbol{R_2}$ is given by $Z^0_{n,1} = 1$ and

$$Z^L_{n,2} = 0,\ I^L_n = d_{n,2} \qquad \text{If } I^{\max}_n \geq d_{n,2}$$
$$Z^L_{n,2} = [d_{n,2} - I^{\max}_n]/u,\ \lceil Z^L_{n,2} \rceil = 1,\ I^L_n = I^{\max}_n \qquad \text{If } I^{\max}_n < d_{n,2}$$

Therefore, $|G^*(P_2) - G^H(P_2)| < \sum_{\forall n} h_n \cdot \min.\{d_{n,2}, I^{\max}_n\}$ holds.

Case 2. $d_{n,1} \leq u < d_{n,1} + d_{n,2}$

In this case, the optimal solution to Problem $\boldsymbol{P_2}$ is given by $Z^*_{n,1} = 1$ and

$$Z^*_{n,2} = 1,\ I^*_n = 0 \text{ regardless } I^{\max}_n < d_{n,2} \text{ or } I^{\max}_n \geq d_{n,2}$$

and the optimal solution to $\boldsymbol{R_2}$ is given by $Z^0_{n,1} = 1$ and

$$I^L_n = u - d_{n,1},\ Z^L_{n,2} = [d_{n,2} - (u - d_{n,1})]/u,\ \lceil Z^L_{n,2} \rceil = 1, \qquad \text{If } I^{\max}_n \geq d_{n,2}$$
$$I^L_n = \min.(u - d_{n,1}, I^{\max}\},\ Z^L_{n,2} = [d_{n,2} - I^L_n],\ \lceil Z^L_{n,2} \rceil = 1, \qquad \text{If } I^{\max}_n < d_{n,2}$$

Since $u - d_{n,1} < d_{n,2}$ *and* $I^L_n \leq d_{n,2}$, $|G^*(P_2) - G^H(P_2)| < \sum_{\forall n} h_n \cdot \min.\{d_{n,2}, I^{\max}_n\}$ again holds. □

As a matter of fact, we can also prove that, regardless $h_n \cdot I^{\max}_n > c^f$ or $h_n \cdot I^{\max}_n \leq c^f$, the claim $|G^*(P_2) - G^H(P_2)| < \sum_{\forall n} h_n \cdot I^{\max}_n$ holds. Lemma 3 implies a potential promise of a partial LP-relaxation based heuristic for solving Problem ***P***. The proposed heuristic, called $\mathbf{H^{LP}}$, works through an iterative process. In each iteration, a planning horizon of *T'* periods, where $T' = T, T-1, T-2, ..., 2, 1,$ is being considered. The integer requirements on $Y_{v,n,1}$ and $Z_{v,1}$ of the given horizon are retained while the integer requirements on all the remaining integer variables are relaxed. This reduces the number of integer variables from $O(VTN)$ to $O(VN)$. By the end of each iteration, all the non-integer solutions, $Y_{v,n,t}$ and $Z_{v,t}$, where $t = 2,3,\ldots,T'$, are disregarded, and the ending inventories $I_{n,t-1}$ and $I_{0,t-1}$ are used as the beginning inventory of the next iteration. The iteration continues until all the integer vessel assignments are obtained.

3. Empirical Observations

In this section, we report our empirical observations on the performance of the proposed heuristic $\mathbf{H^{LP}}$ under various parameter settings, including some real life parameter values. Since the real data are fluctuating and stochastic in nature, we deterministically estimate some fixed parameters, inference to their values in practices and vary some other parameters that tend to be fluctuating in real life.

Refinery/Distribution Port: We consider the integrated operational planning problem involving a single refinery plant (distribution port) and a set of destination ports (customers) ranging from 6 to 12 ports. The length of the planning horizon is 5 time periods, each of which is about 15 days. At the refinery/distribution port, the maximum inventory is limited to 200 thousand tons. In practice, the maximum production rate for each time period varies depending on the crude oil supply. In this study, we consider both real and synthetic maximum production data. In the synthetic case, the maximum production rate is fixed to $\pi^{\max}$ =140, 210, and 280 thousand tons for all period as well a value sampled from $\pi^{\max}$ = U(140,280) in each period. In the real data case, the realistically estimated maximum production rate in each period is provided and the rate varies from period to period.

Chartered Vessels: In practice, most vessel contractors have only a few standard types of vessels. Here, we consider 4 vessel types (small, medium, large, and extra-large), where 10 vessels of each type are available in each period. Their base capacities are fixed parameters of u_v^0=50, 55, 80, and 90 thousand tons for small, medium, large, and extra-large vessels, respectively. Generally, these vessels can carry extra load about 10% of their base capacities; therefore, the maximum loading capacities of these vessels are set to $u_v^{\max} = 1.1 \cdot u_v^0$ thousand tons. The fixed costs for chartering these vessels are \$12,000, \$14,000, \$18,000, and \$20,000, respectively. The rates for overloaded volume that these vessels contractors charge are \$1,000, \$1,000, \$500, and \$500 per additional one thousand tons, respectively.

Destination (Customer) Ports: There are 3 types of destination ports (small, medium, and large) with the maximum inventories of 35, 50, and 65 thousand tons. We generate the synthetic demand data by setting it as a function of maximum production rate, given by $d_{nt} = \pi^{\max} \cdot$ U (0.1,0.2) for small ports, $d_{nt} = \pi^{\max} \cdot$ U (0.15,0.25) for medium ports, and $d_{nt} = \pi^{\max} \cdot$ U (0.2,0.3) for large ports. The average flow rates are 25 thousand tons per day for small ports, 35 thousand tons per day for medium ports, and 50 thousand tons per day for large ports. In general, these destination ports are considered to be customer ports, where the holding costs are substantially lower than the holding cost at the refinery. The ratio of the holding cost at the refinery to a destination port changes overtime depending on many factors (e.g., seasons, demands, contracts). We shall discuss the investigation on the impact of this ratio later in this section.

Berthing, Demurrage, and Travel Times: Each vessel requires a non-negligible berthing and demurrage time to wait for port availability and to set up the oil transfer. In general the berthing and demurrage time is not deterministic; therefore, in this study we vary the berthing and demurrage time $b_{v,n}$ to U(0.5, 3) days. The parameter β is a portion of operating time in each time period dedicated to the oil transfer operation, and (1-β) is a portion of operating time reserved for vessel travel. The value of parameter β realistically can vary depending on the weather and seasonal. We set in this study β to 0.6, 0.7, and 0.8, to reflect different weather conditions for marine transportation.

3.1. The Effect of Increasing Solver Time

We are interested in the error gaps between the heuristic solutions, obtained by applying $\mathbf{H}^{\mathbf{LP}}$, and the optimal solutions obtained by CPLEX. However, due to the combinatorial nature of the problem, verifying the optimality often took CPLEX solver an excessive amount of time in our case. We thus studied the relationship between the search time and the solution quality (see the sampled results in Table 1 below), and noticed that within one-hour CPU time, CPLEX solver (version 9.1) can usually find a feasible solution with a fairly reasonable error gap (well within 3% in our study). Therefore, we used the best solution by CPLEX in one-hour CPU time limit as a baseline in the comparison with the heuristic solutions. The associated mathematical programming problems encountered in the heuristic search process were all solved by the same CPLEX package on a Dell Desktop (PowerEdge 400SC, Pentium4 2.8GHz, 1G RAM).

Table 1. Gaps Between the Optimal Solution and the Best CPLEX Solutions Under the Time Limit.

n	Gap Between Optimal Solution and Best CPLEX Solutions (%)		
	≤ 3600 seconds	≤ 7200 seconds	≤ 10800 seconds
1	2.13%	2.10%	2.06%

2	1.68%	1.65%	1.64%
3	1.70%	1.67%	1.66%
4	2.13%	2.10%	2.06%
5	2.11%	2.08%	2.06%

3.2. The Robustness of Heuristic Approach in What-if Analyses

We then steer our empirical studies toward the robustness of the proposed LP-relaxation based heuristic approach, $\mathbf{H}^{\mathbf{LP}}$. We herein investigate the following 5 scenarios when the real life settings change – reflected on different parameter values. The first scenario is when there is an increasing number of customers and demand. The second scenario is when the maximum production capacity is constant or changes over time. The third scenario is when the ratio of the holding cost at the distribution port to the holding cost at the customer ports changes. The fourth scenario is when the ratio of the shortage penalty to the overload cost changes. The last scenario is when the vessels' travel times change. Since it is not possible to enumerate and test the problem for every setting, we in turn fix some base parameters and vary some other parameters, described in the previous section, in each analysis. The base parameter values of N, π^{max}, h_0/h_n^{avg}, P_n/C_v^{avg}, and β are fixed to 9, 210, 10, UNIF(1,1.5), and 0.7, respectively. Note that each observation is based on the average result from 10 replications.

In the first scenario, we investigate the performance of the proposed LP-relaxation based heuristic approach, $\mathbf{H}^{\mathbf{LP}}$, under different numbers of customer ports (N) while all the other parameters are being fixed at their base values mentioned previously. Table 2 shows the computational performance of the proposed heuristic in terms of the solution values and the required CPU time compared to that by solving Problem ***P*** directly using CPLEX. The value of N is set to 6, 9, and 12 with equal numbers of small, medium, and large ports. The proposed heuristic demonstrates a strong performance while CPLEX fails to find the optimal solution within the specified computational time limit (one-hour) for all the 30 randomly generated test cases here. We can also see that as N increases, the error gaps tend to decrease.

Table 2. Performance of the heuristic against different network sizes (*N*).

Number of Customer Ports *(N)*	CPU Time (Seconds)				Performance GAP		
	CPLEX Optimizer		Partial LP Relaxation		Error Gaps on Total Cost*		
	Average	Maximum	Average	Maximum	Average	Maximum	Std. Dev.
6	3600	3600	0.200	0.218	1.02%	1.07%	0.06%
9	3600	3600	0.227	0.233	0.89%	0.95%	0.05%
12	3600	3600	0.235	0.247	0.84%	0.88%	0.04%

*Gap between the best optimal found with CPLEX so far and the relaxation heuristic

In the second scenario, we consider the case where the maximum production rate, denoted by π^{max}, is a constant for all periods, a sample from a probability distribution for each period, and real data for each period. Table 3 shows the computational performance of the proposed heuristic under various production capacity levels. As we can see in the table, the proposed heuristic has demonstrated a strong computational performance, compared to what CPLEX achieves. It requires a significant less amount of search time while the maximum error gaps are all within 2% over all the 50 test cases.

Table 3. Performance of the Proposed Heuristic against various production capacity levels.

Maximum of Production Capacity (π^{max})	CPU Time (Seconds)				Performance GAP		
	CPLEX Optimizer		Partial LP Relaxation		Error Gaps on Total Cost*		
	Average	Maximum	Average	Maximum	Average	Maximum	Std. Dev.
140	3600	3600	0.201	0.203	1.02%	1.07%	0.04%
210	3600	3600	0.227	0.233	0.89%	0.95%	0.05%
280	3600	3600	0.214	0.223	0.87%	1.12%	0.27%
UNIF(140,280)	3600	3600	0.208	0.219	0.91%	1.09%	0.19%
Real Data	3600	3600	0.210	0.223	1.02%	1.08%	0.09%

*Gap between the best optimal found with CPLEX so far and the relaxation heuristic

In the third scenario, we investigate the case where the ratio between the distribution port's holding cost and the customer ports' holding costs, denoted by h_0/h_n^{avg}, fluctuates. When the ratio is high, the business operations are manufacturing-centric. In other words, timely shifting the product from the refinery site to the customer sites, and thus a more frequent vessel visit, becomes important and highly prioritized. When the ratio is low, the business operations are customer-centric. In other words, an optimal balance between the customer inventories and vessel operations becomes more important. The performance of the proposed heuristic against various holding costs is reported in Table 4. From the table, it is observed that as the ratio increases, the error gap also increases. The reason behind this phenomenon is that under a high h_0/h_n^{avg} ratio, the coordination of vessel scheduling across time periods becomes more critical while the proposed heuristic only focuses on the vessel operations one period at a time.

Table 4. Performance of the heuristic against the holding cost ratios.

Holding Cost Ratio (h_0/h_n^{avg})	CPU Time (Seconds)				Performance GAP		
	CPLEX Optimizer		Partial LP Relaxation		Error Gaps on Total Cost*		
	Average	Maximum	Average	Maximum	Average	Maximum	Std. Dev.
6	3600	3600	0.210	0.219	0.66%	1.59%	0.93%
8	3600	3600	0.217	0.223	0.78%	1.12%	0.48%
10	3600	3600	0.227	0.233	0.89%	0.95%	0.05%
12	3600	3600	0.210	0.227	0.89%	1.08%	0.21%
14	3600	3600	0.206	0.217	0.91%	1.08%	0.18%
20	3600	3600	0.213	0.230	0.90%	1.06%	0.16%

*Gap between the best optimal found with CPLEX so far and the relaxation heuristic

In the fourth scenario, we focus on the drivers for the decision making process in our scheduling. We aim to demonstrate that the proposed heuristic is robust and effective in both make-to-stock and make-to-order environments. The make-to-stock environment is the situation where the shortage penalty is very high and the distribution port always desires to fulfill all the demands by increasing safety stock levels. The make-to-order environment is the situation where the overload cost is very high and the shipment tends to be at its minimum requirement. The ratio of shortage penalty to overload cost, denoted by P_n/C_v^{avg}, represents the change from one environment to other

environment. This ratio varies in different business situations, where balancing supply and demand is a business strategy to minimize the operation costs and maximize the customer satisfaction. The performance of the proposed heuristic against different shortage to overload ratio is reported in Table 5. Overall, the proposed heuristic performs effectively and efficiently in this scenario as well.

Table 5. Performance of the heuristic against the shortage penalty to overload ratios.

Shortage Penalty to Overload Ratio (P_n/C_v^{avg})	CPU Time (Seconds)				Performance GAP		
	CPLEX Optimizer		Partial LP Relaxation		Error Gaps on Total Cost*		
	Average	Maximum	Average	Maximum	Average	Maximum	Std. Dev.
UNIF(0.5,1)	3600	3600	0.202	0.209	0.91%	1.02%	0.11%
UNIF(0.75,1.25)	3600	3600	0.218	0.228	0.97%	1.10%	0.14%
UNIF(1,1.5)	3600	3600	0.227	0.233	0.89%	0.95%	0.05%

*Gap between the best optimal found with CPLEX so far and the relaxation heuristic

In the last scenario, we study the effect of changes in the vessel's travel time along the coastline. In practice, the values of β for different vessels tend to be extremely similar (as opposed to deep sea transportation). Table 6 shows the computational performance of the proposed heuristic under various levels of β value. Again, the proposed heuristic performs very effectively (at about 1% error gap). The reason behind this observation is as follows. As the β value increases, the operation time in each period becomes less restricted; therefore, coordination of vessel scheduling across time periods is then considered to be less critical. This situation is in a favor of the proposed heuristic.

Table 6. Performance of the heuristic against the vessel's operating and travel times.

Proportion of Vessel Operating Time (β)	CPU Time (Seconds)				Performance GAP		
	CPLEX Optimizer		Partial LP Relaxation		Error Gaps on Total Cost*		
	Average	Maximum	Average	Maximum	Average	Maximum	Std. Dev.
0.60	3600	3600	0.210	0.217	1.03%	1.05%	0.04%
0.70	3600	3600	0.227	0.233	0.89%	0.95%	0.05%
0.80	3600	3600	0.229	0.233	0.83%	0.85%	0.01%

*Gap between the best optimal found with CPLEX so far and the relaxation heuristic

4. Conclusion and Future Studies

In this study, we studied a challenging real life supply chain scheduling problem encountered in oil and petrochemical industry, which involves production, inventory, and distribution operations, and requires an integrated scheduling to minimize the total operation cost. We proved the NP-hardness of this problem, and showed that some of its special cases are polynomial time solvable. A partial LP relaxation based heuristic was proposed and demonstrated to have a promising performance under the set of test cases considered in this study. There are several interesting extensions of the work presented here. These include introducing a nonlinear penalty cost for vessels carrying loads beyond their normal capacity, multiple refineries at different locations on the network, and multiple products.

References

[1] F. Baita, W. Ukovich, R. Pesanti, and D. Favaretto (1998), Dynamic Routing-and-Inventory Problems: a Review, *Transportation Research Part A* **32**, 585 – 598.

[2] H.C. Bahl, and L.P. Ritzman (1984), An Integrated Model for Master Scheduling, Lot Sizing and Capacity Requirements Planning, *The Journal of the Operational Research Society* **35**(5), 389 – 399.

[3] J.F. Bard, L. Huang, P. Jaillet, and M. Dror (1998), A Decomposition Approach to the Inventory Routing Problem with Satellite Facilities, *Transportation Science* **32**, 189 – 203.

[4]A. Campbell, M. Savelsbergh, L. Clarke, and A. Kleywegt (1998), The Inventory Routing Problem. In: *Fleet Management and Logistics*, T.G. Crainic and G. Laporte (Eds.), Kluwer Academic Publishers, 95 – 112.

[5] Z.L. Chen, L. Lei, and H. Zhong (2007), Container Vessel Scheduling with Bi-directional Flows, *Operations Research Letters* **35**, 1 – 9.

[6] L. Cheng and M. Duran (2004), Logistics for World-Wide Crude Oil Transportation Using Discrete Event Simulation and Optimal Control, *Computers and Chemical Engineering* **28**, 897 – 911.

[7] M. Christiansen, K. Fagerholt, and D. Ronen (2004), Ship Rand Scheduling: Status and Perspectives, *Transportation Science* **38**(1), 1 – 18.

[8] A. Drexl and A. Kimms (1997), Lot Sizing and Scheduling - Survey and Extensions, *European Journal of Operational Research* **99**(2), 221 – 235.

[9] M. Dror and L. Levy (1986), A Vehicle Routing Improvement Algorithm Comparison of a "Greedy" and a Matching Implementation for Inventory Routing, *Computers and Operations Research* **13(**1), 33 – 45.

[10] K. Fagerholt and M. Christiansen (2000), A Combined Ship Scheduling and Allocation Problem, *Journal of the Operational Research Society* **51**(7), 834 – 842.

[11] N. Geismar, G. Laporte, L. Lei, and C. Sriskandarajah (2007), The Integrated Production and Transportation Scheduling Problem for a Product with a Short Lifespan and Non-instantaneous Transportation Time, *INFORMS Journal on Computing* (in-print).

[12] L. Lei, W. Chaovalitwongse, and S. Bora (2007), Solvable Cases of an Integrated Production and Distribution Problem with Heterogeneous Vessels, Working Paper, Rutgers Center for Supply Chain Management, Rutgers University.

[13] R.Q. Riley (2006), Enterprises: Product Design & Development, The Real Cost of Oil, http://www. rqriley.com/moma5.htm.

[14] D. Ronen (2002), Marine Inventory Routing: Shipments Planning, *Journal of Operational Research Society* **53**, 108 – 114.

Generating Job Schedules for Vessel Operations in a Container Terminal

Thin-Yin Leong, Hoong Chuin Lau
Singapore Management University, 80 Stamford Road Singapore (178902), {tyleong, hclau}@smu.edu.sg

Vessel discharge and load jobs in a container terminal can be divided into sequences, one for each assigned quay crane (QC). A QC sequence lists containers in their desired handling order. The common practice for operators is to make do with just this sequence, without generating a job schedule (i.e. desired start-times of handling each container). Operators who need job schedules generate crude ones by using an average gross crane rate (GCR) gleaned from past experience. For realism, the job schedules should recognize the limit the maximum number of prime-movers and yard cranes that it can deploy. In this paper, we study how job schedules can be efficiently generated that maximizes the GCR (or equivalently minimizes the QC total make-span). With an effective algorithm, resources available to a QC can be adjusted, sharing with other QCs, to ensure that its job sequence can be completed by the target end-time. This would lead to more efficient use of terminal resources and good control on the overall vessel completion time.

Keywords: Container terminal, transport scheduling, flowshop, heuristics, quay crane.

1. Introduction

Top ten ports of the world (such as Singapore) have multiple container terminals that are being operated today by competitive corporations, rather than run as a department of their local government. As such, they are increasingly more concerned about competitiveness, cost and service performance than ever before. They now serve concurrently many vessels alongside and their operation hours are being extended to be non-stop round-the-clock. Together with their growth in size to keep up with global demand, they have become more complex to operate.

To better understand the complexity of operations in the terminals, we will describe a typical vessel operation in its fine detail: A *vessel discharge job* takes 1.5 mins quay crane (QC) cycle to fetch a container from the vessel. A prime-mover (PM) marries up with the QC and the container is loaded onto the PM's trailer. The PM travels for 8 mins transporting the container from the QC to a yard crane (YC). The YC takes the container off the trailer releasing the PM for its next job, and the YC completes its 3.0 mins cycle time to place the container in a yard location. It then moves to a point ready to serve another PM. A *vessel load job* would be similarly constructed with the PM arriving at a YC first and then transporting the container to a QC.

The container discharge and load jobs can be divided into QC sequences, one for each QC assigned to it. A QC sequence lists containers by the container IDs in their desired handling order. Associated with each QC sequence should be a start-time and a target gross crane rate (GCR), i.e. the average number of containers moved in an hour. From these, the target end-time can be deduced.

In this paper, we investigate a real-world problem faced by a container terminal of how job schedules (i.e. desired start-times of handling each container) can be efficiently generated from these sequences. The objective in generating the job schedules is to maximize the GCR or equivalently minimize the QC operation's total make-span. The typical approach currently practised, without generating a schedule, is to make do with just the QC sequence, because generating a feasible schedule proved to be extremely challenging. Another approach is to generate schedules by using an average GCR gleaned from past experience, without considerations of PM and YC equipment constraints. For realism, the job schedules should recognize the limit the maximum number of PMs and YCs that it

can deploy. With an effective algorithm, resources available to a QC can be adjusted, sharing with other QCs, to ensure that its job sequence can be completed by the target end-time. This would thus lead to more efficient use of terminal resources and good control on the overall vessel completion time.

Our problem is very much related to the classical flow-shop scheduling problem. It is in effect a bi-directional flow shop where a job is defined as moving a container from the vessel to a pre-designated yard, or vice versa. Each job comprises 3 tasks – the processing on a QC, the movement by a PM, and the processing by a YC. We are interested in a special constraint that a string of jobs which must be either started or completed in a *strict handling sequence* for one type of machines (i.e. the QCs). This constraint is imposed by the need to discharge or load containers in a certain order from the vessel.

2. Literature

Murty et al (2004 and 2005) and Steenken et al (2004) provide excellent descriptions of container terminal operations. Earlier, operations research studies on container terminal operations have focused on high level strategic design and tactical planning problems: demand forecasting, terminal and berth capacity planning, berth planning, vessel stowage planning and yard planning. There have not been as many studies on operational scheduling and control concerns as these problems tend to be complex, and unless properly handled is best left to practitioners to work out in real-time operational execution. Otherwise, constraint programming and artificial intelligence methods are used. Artificial Intelligence (AI) methods are extremely difficult to implement in real-life and are often avoided by practitioners. So very few are actually implemented into real systems, especially those commercially available.

To support potential automation of the terminals with automated guided vehicles (AGVs) and automatic stacking cranes (ASCs), these complex problems and solution approaches can no longer be avoided. Liu et al (2002) and Bish et al (2001 and 2005) consider the scheduling and dispatching of PMs to support QC operations, addressing specific problems like vehicle routing, job dispatching etc. which container terminal operators cannot force compliance on the PM drivers, and may be they can do a better job finding the best route. Steenken et al 2000 considers the need to coordinate transport and stowage in shipping planning.

Recently, there is a surge of studies (e.g. Zhu and Lim 2004, Ng and Mak 2006, Jung and Kim 2006) on discovering the optimal "crane split" of vessel discharge and load jobs among the QCs for a single vessel. This problem is similar to the production line balancing problem. The output of this would be a sequence for each QC, but these studies largely assume infinite terminal resources to support the QCs. Such problems for the whole container terminal (Moccia et al 2005) are complex and large mixed integer programs, requiring branch-and-cut, agent-based AI and heuristics approaches.

3. Problem Formulation

We now state the notations (refer to Figure 1) for describing the mathematical formulation of our problem. Consider a QC sequence. For the i^{th} container in this sequence, $i = 1,...,n$, let T_i be the job type, S_{ji} be the job start-time and E_{ji} be the job end-time on equipment j, $j \in \{u, x, y\}$ where u denotes QC, x denotes PM and y denotes YC. $T_i \in \{Disc, Load\}$, where $Disc$ denotes discharge and $Load$ denotes load jobs. C_{ui} and C_{yi} respectively are the QC and YC cycle times (which can be simply C_u and C_y if independent of i) and C_{xi} is the PM travel time, computed by dividing the approximate distance between the container's current location to its destination with the average PM speed. There is also an interface time C_f, specified as the minimum time required between a PM arrival to a crane and its departure from that crane, during which a container is placed or removed

from its trailer. In general, all time durations $C_{\cdot\cdot} \geq 0$. By definition, the relationships between the variables are as follows:

$$E_{ji} \equiv S_{ji} + C_{ji}, \quad i = 1,...n,\ j \in \{u, y\} \tag{1}$$

$$S_{xi} \equiv \begin{cases} E_{ui} - C_f = S_{ui} + C_{ui} - C_f, \text{ if } T_i = Disc \\ E_{yi} - C_j = S_{yi} + C_{yi} - C_f, \text{ if } T_i = Load \end{cases}, \quad i = 1,...n \tag{2}$$

$$E_{xi} \equiv \begin{cases} S_{yi} + C_f, \text{ if } T_i = Disc \\ S_{ui} + C_j, \text{ if } T_i = Load \end{cases}, \quad i = 1,...n \tag{3}$$

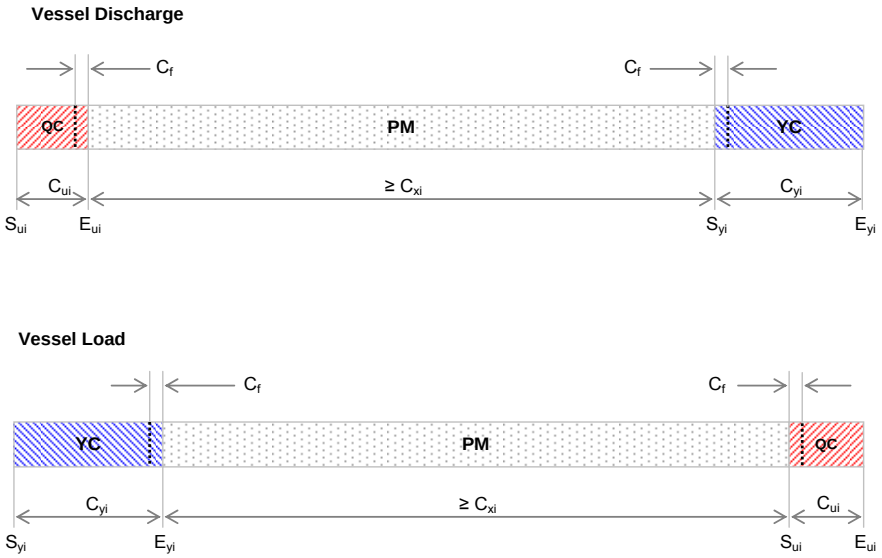


Figure 1. Vessel Discharge and Vessel Load Jobs Time Elements

The problem is to determine S_{ui} and S_{yi} for $i = 1,...,n$, which in turn defines S_{xi} and all the job end-times since the cranes and PMs must work hand-in-hand transferring the containers between them.

The *Job Schedule Generation* Problem is formulated formally as follows:
$(GenSch_{XY})$:

Minimize $Z = E_{un}$ [or Maximize $GCR = n/(E_{un} - S_{u1})$] (4)

subject to

$$S_{ui} \geq E_{ui-1}, \quad i = 2,...n \tag{5}$$

$$n(D_{xi}) \leq X \quad where \quad D_{xi} = \{k : S_{xk} \leq S_{xi} < E_{xk}, k = 1, ..., n\}, \quad i = 1,...n \tag{6}$$

$$n(D_{yi}) \leq Y \quad where \quad D_{yi} = \{k : S_{yk} \leq S_{yi} < E_{yk}, k = 1, ..., n\}, \quad i = 1,...n \tag{7}$$

$$\begin{matrix} S_{yi} - E_{ui} \geq C_{xi}, \text{ if } T_i = Disc \\ S_{ui} - E_{yi} \geq C_{xi}, \text{ if } T_i = Load \end{matrix}, \quad i = 1,...n \tag{8}$$

$$S_{ui} \geq 0 \text{ and } S_{yi} \geq 0 \text{ for } i = 1,...,n. \tag{9}$$

Objective function (4) minimizes the make-span, i.e. the completion time of the QC sequence. S_{u1} is the start-time of the first QC job and also the sequence start-time. Without loss of generality, we let $S_{u1} = 0$. Constraint (5) specifies that the QC handling order and its capacity are not violated. Constraints (6) and (7) ensure respectively that the number of PMs and YCs deployed at the start-time of any job i do not exceed the available X prime movers and Y yard cranes. Since increase in equipment deployed happens only at the start-times of PM and YC jobs, constraints (6) and (7) will ensure that limits X and Y are not exceeded for the whole duration of working the QC sequence. Constraint (8) states that the separation time between crane handlings at both ends of a PM travel must be at least as long as the travel time so that it is physically possible to execute. Constraint (9) is the non-negativity constraint on the crane start-times. The optimal schedule of $GenSch_{XY}$ will be referred to as $SchOpt$.

One decision problem associated with $GenSch_{XY}$ is: Is there a schedule with make-span no greater than a given value Z using no more than X PMs (or no more than Y YCs, or both)? Note that if there is no restriction on the QC handling sequence, this problem is *NP-complete*, since the Bin-Packing problem can be trivially reduced to it by treating each PM as a bin and Z as the bin capacity. Interestingly, when the QC sequence is strictly imposed, this problem can be nicely solved by a greedy algorithm, as we will show in the next section.

4. Solution Approaches

It can be observed that good schedules for $GenSch_{XY}$ should pack the jobs tightly, with minimal time gaps between the end of one QC job and the start of the next QC job. Hence, one approach for generating a job schedule is to pack the jobs one after another, so long as they follow the QC handling sequence constraint. This algorithm (denoted as $Sch1$) would generate the highest possible GCR, but violates the PM and YC availability constraints.

In the case where we do not permit the PM jobs to wait, we can simplify by replacing definitions (2) and (3) with (10) and (11) and adding constraint (12):

$$S_{xi} \equiv \begin{cases} S_{ui} + C_{ui} - C_f, \; if \; T_i = Disc \\ S_{ui} - C_{xi} - C_f, \; if \; T_i = Load \end{cases}, \quad i = 1,...n \tag{10}$$

$$E_{xi} \equiv S_{xi} + C_{xi} + 2C_f, \quad i = 1,...n \tag{11}$$

$$S_{yi} \equiv \begin{cases} S_{ui} + C_{ui} + C_{xi}, \; if \; T_i = Disc \\ S_{ui} - C_{xi} - C_{yi}, \; if \; T_i = Load \end{cases}, \quad i = 1,...n \tag{12}$$

Invoking constraints (10) and (11) together is the same as changing the inequality in (8) to equality, making the constraint binding. This resulting problem will be referred to as $GenSch2_{XY}$, under which only $S_{ui}, i = 2, ..., n$ is a decision variable; formerly a decision variable, $S_{yi}, i = 1, ..., n$ is now a dependent variable. That is, we need only find the QC job start-times, as the rest of the start-times can be easily derived by adding or subtracting the cycle, travel and interface times.

$GenSch2_{X\infty}$ (and hence $GenSch_{X\infty}$) can be solved optimally using a greedy algorithm, both for a given set of purely discharge jobs or purely load jobs. In the following, we will present our algorithm for the case of discharge jobs, followed by load jobs. It turns out that the latter algorithm is identical to the former, in reverse time sequence, as we will demonstrate.

Scheduling Discharge Jobs: We assign jobs in increasing QC handling order. For each job *i*, we assign its start time to be the earliest time that does not violate the PM and YC capacities. In other words, S_{ui} is the earliest time such that (1) S_{ui} is no earlier than the end time of job *i*-1; (2) S_{xi} is the time when a PM is available; and (3) S_{yi} is the time when a YC is available. Let this algorithm be denoted as *Sch-Discharge*.

Scheduling Load Jobs: Again, we assign jobs in increasing QC handling order. The trick here is to think of load jobs as discharge jobs in a *reverse* time sequence – so we perform scheduling in a reverse time sequence where the QC task becomes the first instead of the last task. In doing so, we shift the burden to maintain QC handling order at the *end* to the *beginning*, and hence the same algorithm in *Sch-Discharge* may be applied. After scheduling, we simply reverse again the start and end times of each job. This trick has been shown to work for the problem discussed in Bish et al (2005). Let this algorithm be denoted as *Sch-Load*.

4.1 Analysis

Both *Sch-Discharge* and *Sch-Load* can be implemented to solve $GenSch_{X\infty}$ in worst-case time complexity of O(n log X). For each job, it suffices to determine when a PM would be available. If we maintain a heap of size X which contains the completion timings of each PM, then obtaining the earliest completion time of a PM and updating the new completion time (as a result of a job assignment) is equivalent to performing a *decreaseKey* operation on the respective heap, which takes logarithmic time. This also means that if X is constant, then we have a linear time greedy algorithm to solve $GenSch_{X\infty}$. For the general case $GenSch2_{XY}$, the situation is slightly tricky since there could be cases when a PM is available but a YC is not, and vice versa. To ensure that we find the earliest available time slot to assign a job, we need to scan from the earliest available time of a PM forward until the time when a YC is available. Typically, since the YC cycle is short, this scan is negligible

In the following, we will prove that both *Sch-Discharge* and *Sch-Load* are optimal for restricted cases.

Proposition 1: *Sch-Discharge* is optimal to $GenSch2_{X\infty}$ if all jobs are discharge jobs.

Proof: This can be shown by an exchange argument as follows. Denote the schedule obtained by *Sch-Discharge* as *Sch*. Suppose that there exists an optimal schedule *SchOpt* that is *not* the same as *Sch.* Consider each job assignment in *SchOpt* from 1 to n in that order. Let job i be the first job whose start time is different from *Sch*. Let the start times of this job be denoted S_{ui} and S_{ui}^{OPT} respectively. Since the start-time for job i in *Sch* is the earliest possible, we have $S_{ui} < S_{ui}^{OPT}$. Now shift the start-time of job i in *SchOpt* backward from S_{ui}^{OPT} to S_{ui}. Note that this will not introduce infeasibility with the jobs before i, since *Sch* is a feasible schedule. Clearly, shifting backwards will also not affect the feasibility of the jobs after i. Furthermore, the resulting make-span will either remain or improve. Following this argument inductively, we can convert *SchOpt* to *Sch* without affecting the make-span. Hence, *Sch* is also an optimal schedule. □

Corollary: *Sch-Load* is optimal to $GenSch2_{X\infty}$ if all jobs are load jobs.

Proof: Again, a simple argument as above can show this.

Unfortunately, the above algorithm is not optimal for $GenSch2_{XY}$ in general. Consider the following counter example comprising a sequence of 6 pure discharge jobs where the QC and YC cycle times are 1 and 2 respectively, PM travel times are 10, 9, 10, 7, 6, 20 respectively, and capacities X=6 and Y=2. In this case, *Sch-Discharge* produces a make-span of 32 while the optimal make-span is 30.

Thus far, we have considered algorithms for purely discharge or purely load jobs. If we are given a *mixed* sequence of discharge followed by load jobs, then it is technically cumbersome to switch between *Sch-Discharge* and *Sch-Load.* Furthermore, this presents the additional challenge to splice the solutions together in a manner that can exploit savings at the cross-over time region between discharge and load jobs. For instance, some discharge jobs might be deferred to allow some load jobs to start earlier, so that the QC will not need to wait for the load jobs to arrive upon completion of all discharge jobs.

In the following we will consider a new single-pass forward algorithm to handle mixed job sequences. This algorithm is simple, efficient and effective (see Experiments section). Unfortunately, it does not preserve the optimality property presented in Propositions 1 and 2. It will however do so if we consider a *restricted* version of the $GenSch2_{XY}$ problem. We define function MAX_m to return the m^{th} largest value, and replace constraints (6) and (7) by (13) and (14):

$$S_{xi} \geq MAX_X\left(E_{xk} : k = 1,...,i-1\right) \quad i = 2,...n \tag{13}$$

$$S_{yi} \geq MAX_Y\left(E_{yk} : k = 1,...,i-1\right) \quad i = 2,...n \tag{14}$$

$GenSch2_{XY}$ modified as above will be referred to as $GenSch3_{XY}$. It is clear that (13) $\Rightarrow$ (6) and (14) $\Rightarrow$ (7). Hence, feasible solutions to $GenSch3_{XY}$ provide feasible and upper bound solutions to $GenSch_{XY}$.

The following greedy algorithm $Sch3_{XY}$ is proposed: Start sequentially from job $i = 2$ to find S_{ui} and then compute the rest using (1), (10), (11) and (12). Using the property of constraints (13) and (14), the feasible solution for each job i can be found by shifting only job i forward so that its PM job start-time is not earlier than the X^{th} largest end-time from among all the PM jobs preceding it and its YC job's start-time is not earlier than the Y^{th} largest end-time from among all the YC jobs preceding it. Again, we seek to restore feasibility with least delay to job i. Both algorithms generate the job start-times for QC, PM and YC jobs in a single pass, without violating the handling order of QC jobs or exceeding X and Y (the PM and YC availability limits). It can be shown that $Sch3_{XY}$ is optimal for $GenSch3_{XY}$.

5. Experiments and Discussions

In this section, we report computational experiments to test the efficiency of the algorithms and quality of results measured in real-world terms. Extensive experimental results that compare our heuristics with other heuristics and exact MIP models have been reported in a separate paper (see Zhao et al 2007).

A total of 432 realistic test cases were generated for the following data values (all times in minutes): $C_f = 0.25, C_u \in \{1.0, 1.5, 2.0\}, C_y \in \{2.3, 3.0, 3.5\}$, $C_{xi} \in \{8 \pm 4, 10 \pm 5, 12 \pm 6\}$ (uniformly-distributed), $X \in \{4, 5, 6, 7\}$ and $Y \in \{2, 3, 4, 5\}$. For each test case, a QC job sequence of 100 jobs is randomly generated, with 50 discharge jobs followed immediately by 50 load jobs. We compute the following results: average (approx.) and maximum PMs and YCs used under $Sch3_{XY}$, maximum number YC used and number of times more than Y YCs are used under $Sch3_{X\infty}$, and GCR under $SchOpt$, $Sch3_{XY}$ and $Sch3_{X\infty}$.

Results of the computational experiments are as follows:

(1) $Sch3_{XY}$ is near-optimal with $(GCR^{Sch3_{XY}} - GCR^{SchOpt}) / GCR^{SchOpt}$ = -0.3% average and -3.6% maximum. *SchOpt* takes 5sec to 43sec to compute, averaging at 22sec. With $GCR^{Sch3_{XY}} \approx GCR^{SchOPT}$ and $Sch3_{XY}$ running in sub-second time, we conclude that in real life, constraints (10) to (12) are generally reasonable assumptions to make and $Sch3_{XY}$ offers a very practical near-optimal approach.

(2) $Sch3_{X\infty}$ was found to be better than optimal: $(GCR^{Sch3_{X\infty}} - GCR^{SchOpt}) / GCR^{SchOpt}$ = 4.4% average and 33.5% maximum. It is of course infeasible relative to the constraint on Y YCs. However, the number of times Y was violated was found to be moderate: 11 jobs on average and 55 jobs maximum out of 100.

Using our proposed scheme for generating job schedules $Sch3_{XY}$, we are able to examine the relationship between GCR and various PM/QC and YC/QC available ratios, which is shown in Figure 2. The GCR increases with PM/QC rising linearly and then saturates at different levels that increases with YC/QC. This is consistently with what is observed in actual terminal operations, though they usually track the relationship to the deployed ratios rather than the availability limits.

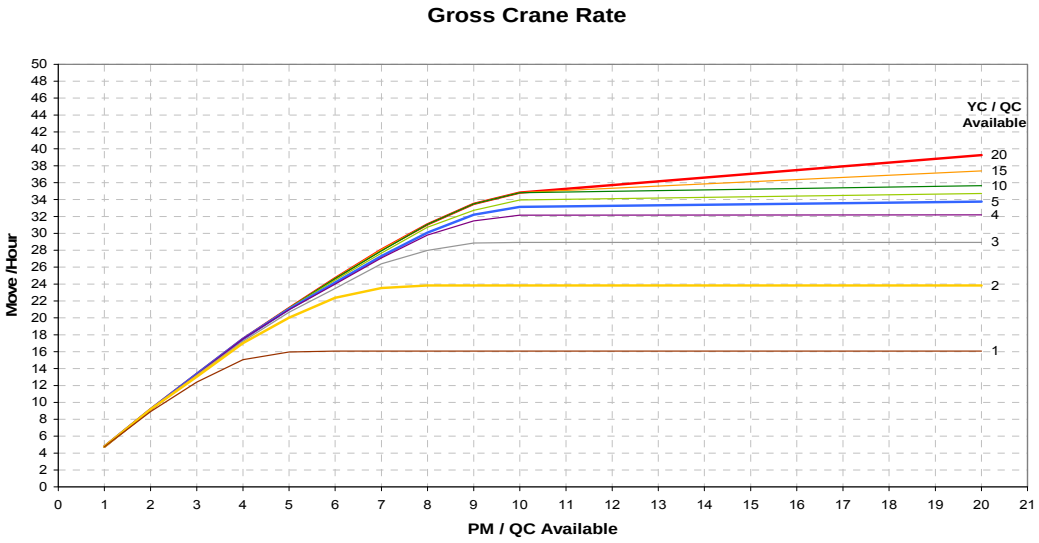


Figure 2: Gross Crane Rate vs PM and YC equipment available

6. Concluding Remarks

The most critical resource in the container terminals is usually not the YC: YC holding is typically almost four times that of QC. The liberal relative quantities of YC is mostly to cater for the fact that YCs are physically distributed over a very large area and they are not mobile enough to be quickly moved from one location to another location. Hence, the key issue is not YC availability but activating the right sub-set of YCs. The terminals can usually do with more PMs since QC is the most expensive resource of the three, and thus should not be slowed down unnecessarily by the lack of YCs or PMs. But having too many PMs without highly sophisticated coordination means would only lead to queues in the wrong places and massive traffic congestion. Hence the controlling variable in container terminal productivity is usually PM availability. For this reason, we can safely focus on $GenSch_{X\infty}$ rather than $GenSch_{\infty Y}$.

We observe that the number of YC deployed does not go frequently above reasonable Y limits when the schedule is generated using $Sch3_{X\infty}$. In fact, the number YCs used are often below Y, i.e.

constraint (7) is usually non-binding. This suggests that we may not need to constraint YCs and let it be an outcome of the PM/QC availability instead. Additional YCs required, beyond base availability given, may be marshaled for blocks of time from other QCs. Conversely, this QC can share its YCs with other QCs when they need them for short periods of time. Therefore, the result exploited in a multiple QCs situation would result in higher GCRs for all QCs, with no additional investment in YCs by the container terminal.

Without loss of generality, we have made simplifying assumptions in our study, e.g. each PM carries only one container and only one yard location is visited per PM trip. Further operational refinements are needed for deploying the proposed algorithm in container terminal operations, which include these and many other practical details. These additions however would not compromise or alter the nature of results reported here. As an example, we may replace PMs in our model with straddle carriers (SCs), the alternative mover-stacker used in terminal operations. SC terminal operations has the advantages that no YCs are needed since SCs can stack containers in the yard, and QCs and SCs do not have to wait for each other as they can put and pickup containers from the wharf side, i.e. their operations are de-synchronized within reasonable limits. Our results will still apply.

In this paper, we considered job scheduling for a single QC. In a continuing work, we study a decentralized multi-QC coordination problem where resources (PM, YCs) are shared. There, we employ a combinatorial auction mechanism to broker the utilization of shared resources to conflicting demands arising from multiple QCs. The algorithm proposed in this paper provides the base algorithm for individual QCs (bidders) to generate their optimal resource bundle bids to be submitted to the auctioneer.

References

[1] Bish, Leong, Li, Ng and Simchi-Levi (2001), Analysis of a New Vehicle Scheduling and Location Problem, *Naval Research Logistics* **48**(5), 363 – 385.

[2] Bish, Chen, Leong, Nelson, Ng and Simchi-Levi (2005), Dispatching Vehicles in a Mega Container Terminal, *OR Spectrum* **27**(4), 491 – 506.

[3] Jung and Kim (2006), Load scheduling for multiple quay cranes in port container terminals, *Journal of Intelligent Manufacturing* **17**(4), 479 – 492.

[4] Liu, Jula and Ioannou (2002), Design, simulation, and evaluation of automated container, *IEEE Transactions on Intelligent Transportation Systems* **3**(1), 12 – 26.

[5] Moccia, Cordeau, Gaudioso and Laporte (2005), A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal, *Naval Research Logistics* **53**(1), 45 – 59.

[6] Murty, Liu, Wan and Linn (2004), A decision support system for operations in a container terminal, *Decision Support Systems* **39**(3), 309 – 332.

[7] Murty, Wan, Liu, Tseng, Leung, Lai and Chiu (2005), Hong kong International Terminals Gains Elastic Capacity Using a Data-Intensive Decision-Support System, *Interfaces* **35**(1), 61 –75.

[8] Ng and Mak (2006), Quay crane scheduling in container terminals, *Engg Optimization* **38**(6), 723 –737.

[9] Steenken, Voβ and Stahlbock (2004), Container terminal operations and operations research – a classification and literature review, *OR Spectrum* **26**, 3 – 49.

[10] Zhao, Leong, Ge, and Lau (2007), Bidirectional Flow Shop Scheduling with Multi-Machine Capacity and Critical Operation Sequencing, *Proc. 22nd IEEE International Symp. on Intelligent Control*.

[11] Zhu and Lim (2004), Crane Scheduling with Spatial Constraints: Mathematical Model and Solving Approaches, *Proc. 8th International Symp. AI and Math*, Florida, USA.

Scheduling Tests on Vehicle Prototypes using Constraint Programming

Kamol Limtanyakul, Uwe Schwiegelshohn
Robotics Research Institute, Dortmund University, 44227 Dortmund, Germany,
{kamol.limtanyakul, uwe.schwiegelshohn}@udo.edu

In this paper, we address the problem of scheduling tests on vehicle prototypes using Constraint Programming. The problem originates in the automotive industry where a manufacturer must perform several hundreds of tests on prototypes before starting mass production of a vehicle. Each test must be allocated to an appropriate prototype with respect to hardware requirements. Further, it is necessary to observe various test dependencies. The manufacturer is interested in reducing the number of prototypes to save test costs and in minimizing the makespan in order to start the production as early as possible.

We formulate the problem using Constraint Programming with the makespan being the primary objective. The number of prototypes is a parameter which must be specified before starting the optimization procedure. Using exemplary input data given from a car manufacturer, we solve this single objective optimization problem several times for different values of our parameter to determine the relation between the number of prototypes and the corresponding makespan. We are able to either find the optimal solution or at least a good feasible solution within a reasonable computational time even for our largest problem size comprising about five hundreds tests and more than one hundred prototypes.

Keywords: Applications, Constraint Logic Programming, Machine Scheduling

1 Introduction

In the automobile industry, a manufacturer must typically perform several hundred tests on vehicle prototypes before a vehicle series goes into production. These prototypes are handmade and expensive (between 0.5 to 1.5 Million Euro each). Most prototypes can be used for several tests if the tests are arranged in an appropriate order. For instance, any crash test must obviously be the last test on this prototype. This leads to a parallel machine scheduling problem with prototypes and tests being machines and jobs, respectively. Therefore, we must determine the allocation of tests to prototypes and the appropriate sequence for all tests allocated to a prototype. Ignoring cost differences between different versions of prototypes, the problem has two major objectives as we want to minimize the number of prototypes required to perform all tests and the makespan of the schedule in order to start the production as early as possible.

Further, there are many additional constraints. For instance, each prototype is a combination of different kinds of components, like engine or gear box. Due to various incompatibilities between those components, this produces in practice up to 600 possible variants with different costs and production times. Of course, we can only allocate a test to a prototype if this prototype satisfies the component requirements of this test. For instance, while a prototype with a gasoline engine is not suited to execute a diesel engine test it does not matter which kind of engine is used to perform a brake system test. However note that we do not yet consider how to configure a proper prototype variant as this requires further technical information to match various components, for instance. The list of possible variants is previously determined by the manufacturer and can be

used several times. Our scheduling algorithm must find the minimum number of prototypes and their corresponding variants from the given list to ensure that all tests can be performed.

We must also take into account temporal restrictions like release dates and due dates. For instance, consider a driving test in winter. As the complete testing process typically takes more than one year, we can obtain appropriate conditions for the driving test by using appropriate release and due dates. Also the manufacturer may use due dates to ensure that some critical tests are completed early enough to allow a repetition in case of a failure.

Most companies have a special shop that produces the prototypes. Due to the limited capacity of this shop, the prototypes are sequentially manufactured resulting in an availability time for each prototype. Following the manufacturing, a prototype requires an initial set-up process whose duration depends on the selected variant of the prototype. For instance, a prototype for a corrosion test must be painted while this is not necessary for executing a crash test. Hence, the scheduler must ensure that tests can start only when the assigned prototype is constructed and sufficiently prepared for the test. A valid schedule must obey further constraints. For instance, a brake system test must be completed before starting a long-run test. Further, the same prototype cannot be used for two long-run tests. Finally, a brake system test and a suspension system test must be executed on the same prototype in order to verify the co-ordination of both systems together.

We use Constraint Programming (CP) to formulate our scheduling problem. Further, we realize that the number of prototypes is discrete while the makespan is (almost) continuous. Therefore, we apply the classical approach of parameterization, that is, we fix the number of prototypes and minimize the makespan with respect to all constraints. Initially, we assume a large number of prototypes such that the makespan only depends on the temporal constraints. Then we repeatedly reduce the number of prototypes and solve the problem again. This produces a relation between the number of prototypes and the makespan.

This paper is organized as follows. Section 2 provides background information about previous work and other relevant projects. Then we formally define our problem and introduce appropriate notations in Section 3. The CP model of our problem is explained in Section 4. The computational results in Section 5 show the capability of our method to handle various sizes of this problem within a reasonable time. Finally, we end with a brief conclusion and an outline of future work.

2 Previous Work

To solve this problem, Scheffermann et al. [9] have developed a heuristic approach based on specific problem knowledge and on some intuitive ideas. They improve the quality of the result by applying statistical methods to adjust tuning parameters. This approach is used as a support tool to help the manufacturer make decisions in a real working environment. However, it is still difficult to find the appropriate parameters that lead to a good result. A deterministic method may be more suitable.

Mixed-Integer Linear Programming (MILP) is a well known method to solve many combinatorial optimization problems. Although it has been used for several decades, many researchers report that frequently MILP alone is not an effective tool to solve large-scale scheduling problems for parallel machines with release and due dates. For instance, Hooker [4] and Sadykov and Wolsey [8] compared the performance of several methods, including the MILP approach, to minimize makespan and allocation cost. Their results showed that MILP can handle small problems well but that it is not appropriate for cases dealing with many jobs like our problem.

Constraint Programming (CP) is an alternative deterministic approach, see Baptiste et al. [1]. It has originally been developed for computer science applications. CP consists of two main parts:

a modeling part which generates a set of constraints that must be satisfied and a search part that describes how to search for solutions. CP will start by searching for a solution that satisfies all constraints. This part is known as Constraint Satisfaction Problem. Basically, CP applies a mechanism called constraint propagation to reduce the domain of variables. However, that is usually not enough to obtain a feasible solution. The solver must still enumerate through the remaining set of variables. Further, the reduction and the search parts must simultaneously interact to determine a feasible solution. In the next step, CP adds a new constraint to ensure that the new objective is strictly better than the current value. As a result, the solution will be improved until it reaches the optimal value.

CP is superior to MILP in expressing constraints that are not limited to linear inequalities. Hence, some kinds of constraints, like disjunctive constraints, can be formulated more naturally and solved more effectively. Also, machine eligibility constraints can obviously be represented with the help of logic expressions, like if...else, instead of using many complex linear inequalities.

CP can solve a problem efficiently if constraints propagate well and tighten the objective value. Therefore, minimization of the makespan is well suited as the last completion time of all job directly leads to a lower bound of the objective value. For minimization of a sum of the set-up times or the allocation cost, any bounds of the cost function do not indicate directly which term of the sum should be reduced, see Danna and Pape [3].

In addition to the previous work [9], there are a few similar projects related to vehicle tests in the automotive industry. Zakarian [10] developed an analysis model and a decision support tool to evaluate the performance of product validation and test plans for General Motors Truck Groups. His work concentrates on stochastic scheduling. He models uncertainties associated with processing times of tests and product failures in order to determine the trade-off between the number of vehicles used in the validation plan and percentage of completed tests. First, several heuristic rules are applied to generate initial schedules based on the probability function. Then, the performance of obtained schedules is measured by a Markov model and simulation approaches.

Lockledge et al. [6] applied a multi-stage mathematical programming model to optimize the fleet of prototypes for Ford Motor Company. In the first stage, they determine the number of required variants subject to component requirements of all tests. Then, the second model solves the problem for a minimum number of cars of each variant such that all tests can be executed before their due dates. It is possible to apply MILP in this case because there are few different values of due dates instead of arbitrary values. The problem reduces to an assignment problem of tests to one of these time slots. This obviously simplifies the model and reduces the number of integer variables. This method is not applicable to our case as the values of our due dates are widely spread over the whole time scale. Moreover, the model will become rather complicated, if we want to consider other temporal constraints like release dates and availability times.

Bartels and Zimmermann [2] used time-indexed decision variables to formulate a MILP model which can directly minimize the number of built prototypes. As only small size problems can be solved optimally, they further developed an heuristic approach to cope with real-life problems. Tests are successively scheduled according to the given priority while the number of prototypes only increases when there is still a test which cannot be allocated on the existing prototypes.

3 Problem Description and Notations

After informally explaining the problem in the Section 1, we now introduce the notations used in the rest of the paper. These notations are based on Pinedo [7] whenever possible.

V, I, and J are the sets of prototype variants, prototypes, and tests, respectively, with $|V| = l$,

$|I| = m$, and $|J| = n$. Each prototype $i \in I$ corresponds to variant $v_i \in V$. $M_j \subseteq V$ is the set of prototype variants that can perform test $j \in J$.

Each test $j \in J$ has a processing time p_j, a release date r_j, and a due date d_j. As all tests are executed without interruption, the completion time C_j of test j must obey $r_j + p_j \leq C_j \leq d_j$. Further, we consider the following relations between two different tests $j, k \in J$:

- $j \prec k$ iff test j must be completed before test k is started.
- $j \sim k$ iff tests j and k must be executed on the same prototype.
- $j \asymp k$ iff tests j and k must not be executed on the same prototype.

Finally, $J_{Last} \subseteq J$ is the set of tests that must not be followed by any other test on the same prototype.

Further, we must formulate the availability time of a prototype. As already mentioned, we use a simple model for this purpose. This model considers manufacturing time of a prototype, the set-up time of a prototype variant and a potential delivery delay of a component of this prototype. In general, prototype $i \in I$ is not available before time a_i which is independent of v_i. An additional time s_v is required to set-up variant $v \in V$. Occasionally, there may be a delay for the delivery of a component such that this delay cannot be compensated within prototype manufacturing. In this case, we assume that the delay time y_v of prototype variant v dominates the manufacturing of the prototype and includes the integration of this component into the already existing prototype. Therefore, a test cannot be executed on prototype i before time $\max\{a_i + s_{v_i}, y_{v_i}\}$. Finally, the objective of this scheduling problem is to minimize the makespan $C_{max} = \max_{j \in J}\{C_j\}$.

To employ the common notation of scheduling problems, we consider prototypes and tests as machines and jobs, respectively. Using the standard scheduling framework, this problem can be classified as an extension of scheduling identical machines in parallel with machine eligibility constraints since a job can be performed with constant processing time on any machine (prototype) which has suitable characteristics (variant). Therefore, the notation $P_m|r_j, d_j, M_j, prec|C_{max}$ represents our problem. However, we must be aware that this notation does not include all facets of the problem like the selection of a variant for each machine (prototype). This selection determines the machine eligibility restrictions, M_j. Further, we must consider application-specific conditions like performing two tests on the same prototype ($j \sim k$). The problem is NP-complete in the strong sense, as it is a generalization of $P_m|prec|C_{max}$, see Pinedo [7].

4 CP Formulation

It is quite common in CP to define a resource constraint for a single machine in a general scheduling problem by using the term **cumulative**$(\vec{t}, \vec{p}, \vec{c}, C)$, where $\vec{t} = [t_1, .., t_n]$, $\vec{p} = [p_1, .., p_n]$, $\vec{c} = [c_1, .., c_n]$ are the vectors of starting time, processing time, consumption rate of jobs, respectively, and C is the capacity of the machine. The constraint is satisfied if the condition $\sum_{j \in J_t} c_j \leq C$ holds for all time instances t in the valid time frame, where $J_t = \{j \in J | t_j \leq t \leq t_j + p_j\}$ is the set of tasks that are in process at time t. Therefore, the total consumption of all jobs $j \in J_t$ does not exceed the capacity C.

In this model, we consider each prototype i as an unary resource with capacity $C = 1$ as it can perform at most one test at a time. Its corresponding variant is represented by variable $v_i \in V$ $\forall i \in I$. There are also other possibilities to model the resources of this problem. For instance, we may consider a group of prototypes with the same variant as a single cumulative resource with

unknown capacity (the number of prototypes). However, this approach requires CP solvers which can directly define the capacity of a cumulative resource by using a variable.

Further, we have $c_j = 1, \forall j \in J$ as each test requires a single prototype. The starting time t_j of a test j must be in the interval $[r_j, ..., d_j - p_j]$ to obey the release date and due date constraints. Finally, $x_j \in I, \forall j \in J$ represents the allocation of test j to a prototype. This leads to the following CP formulation:

$$\text{Minimize} \quad C_{max}$$

subject to

$$C_{max} \geq t_j + p_j, \qquad \forall j \in J \tag{1}$$
$$t_j \geq r_j, \qquad \forall j \in J \tag{2}$$
$$t_j + p_j \leq d_j, \qquad \forall j \in J \tag{3}$$
$$\text{cumulative}(t_j | x_j = i, p_j | x_j = i, c_j = 1 | x_j = i, 1), \forall i \in I \tag{4}$$
$$v_i \notin M_j \Rightarrow x_j \neq i, \qquad \forall i \in I,\ \forall j \in J \tag{5}$$
$$x_j = i \Rightarrow t_j \geq a_i + s_{v_i}, \qquad \forall i \in I,\ \forall j \in J \tag{6}$$
$$x_j = i \Rightarrow t_j \geq y_{v_i}, \qquad \forall i \in I,\ \forall j \in J \tag{7}$$
$$t_j + p_j \leq t_k, \qquad \forall j \prec k,\ j,k \in J \tag{8}$$
$$x_j = x_k, \qquad \forall j \sim k,\ j,k \in J \tag{9}$$
$$x_j \neq x_k, \qquad \forall j \asymp k,\ j,k \in J \tag{10}$$
$$x_j = x_k \Rightarrow\ t_j \geq t_k + p_k, \qquad \forall j \in J_{\text{Last}}, \forall k \in J, j \neq k\ . \tag{11}$$

Constraint (1) ensures that the makespan is not smaller than the completion time of any test. Constraint (2) and Constraint (3) state that tests must be performed between their respective release and due dates. Constraint (4) is a resource constraint with the term $(t_j | x_j = i)$ denoting the tuple of starting times for tests that are assigned to prototype i. The constraint prohibits the concurrent execution of two jobs on the same machine. Constraint (5) prevents that a test is assigned to a prototype whose variant does not belong to the eligibility set of the test.

Constraint (6) and Constraint (7) ensure that a test on machine i can neither start before the availability of the prototype according to our model. Remember that both set-up time s_{v_i} and component available time y_{v_i} depend on a value of variable v_i, which is unknown before optimization. It is another benefit of the CP approach that variables can index parameter arrays.

Precedence constraints are represented in Constraint (8). Constraint (9) makes sure that any pair of tests (j, k) with $j \sim k$ will be executed on the same machine. Further, tests j and k with $j \asymp k$ must be processed on different machines which is achieved by Constraint (10). Finally, Constraint (11) ensures that test $j \in J_{Last}$ will be the last job executed on the machine to which it is allocated. Some of the constraints are very specific to our problem. But due to its flexibility, the CP model can be adapted rather easily to problems using a similar parallel machine environment.

5 Computational Results

To demonstrate our approach, we apply the CP model together with data obtained from a real-life test scenario. There are four data sets of different sizes from about 40 tests to almost 500 tests. Table 1 shows other details of those data sets, like the number of variants and the number of tests

with precedence constraints. It can be seen that up to 10% of all tests may be subject to special constraints. In the large data set involving several hundred tests, the impact of those constraints may be considerable. Therefore, it is difficult to later verify and correct a schedule that has been obtained by neglecting these requirements.

Table 1: Detailed information about the given data sets.

Data	n	l	$\lvert\{j \in J \lvert j \prec k\}\rvert$	$\lvert\{j,k \in J \lvert j \sim k\}\rvert$	$\lvert\{j,k \in J \lvert j \asymp k\}\rvert$	$\lvert J_{Last}\rvert$
1	41	38	1	2	2	1
2	100	3	1	2	2	-
3	231	9	3	6	6	3
4	486	663	1	54	2	1

We use OPL Studio 3.7 [5] to formulate and solve the CP model introduced in Section 4. In addition to the standard CP functions, OPL also provides a specific scheduler module that includes several functions for solving scheduling problems. Further, it has good constraint propagation techniques and efficient searching algorithms. The numbers of variables and constraints presented in Table 2 correspond to the model formulated in OPL. However, these numbers are not directly related to the difficulty of the optimization problem. We only provide them to give an impression of the problem size and the their variations during the calculations. All computations were performed on a Pentium IV, 3.0 GHz, and 1 GB main memory.

Table 2 shows the computational results. For each data set and a given number of prototypes, we minimize the makespan. Initially, we may set m to a large value which means we try to construct as many prototypes as time allows in order to accommodate all tests. In some cases, this procedure does not lead to a feasible solution. For instance, prototypes may be available too late if there are many tests with very early due dates. However, it can suggest the conflicts should arise from temporal constraints rather than from a limited number of prototypes.

After obtained the results, we reduce the number of prototypes step-by-step to see how it affects to the makespan. This procedure is repeated until it becomes infeasible to find a solution or the calculation cannot terminate within a given time limit. We specify this time limit to be 1 hour for Data 1–3 and 6 hours for Data 4. For example in Data 1, we start from $m = 14$ and obtain a makespan of 330 days. If we restrict ourselves to only 5 prototypes we still get the same optimal makespan. However, if we set $m = 4$ the problem becomes infeasible as it has not enough prototypes to execute all tests with respect to all constraints.

Provided there is enough time, the solver either achieves an optimal solution for this number of prototypes or proves that the problem is infeasible. But the required computation time grows quickly when the problem size becomes larger. Due to limitations of computation time in practice, the solver may not be able to prove the infeasibility of the problem. Even if it finds a feasible solution we cannot be sure about its optimality as long as there are still unexplored nodes in the search tree. The results show that for all data sets, the optimal solution can be achieved when the number of given prototype is large enough. If the number of available prototypes decreases it is getting more difficult to find any feasible solution or prove optimality, although the numbers of variables and constraints decrease. However, if the number of prototypes is sufficiently reduced the solver can again prove infeasibility within the limit time because the search space has shrunk substantially. For instance, the calculation for Data 3 shows that the optimal solution can be obtained when $m = 20$. For $m = 18$ or $m = 19$, feasible solutions with the slight increase of the makespan are found after one hour of computation time. But, it has an unknown gap in the range

of $9 \leq m \leq 17$ prototypes. We cannot determine whether the problem can be solved or not for a number of this set.

Further, it should be noted that an optimal value of the makespan in step m can be regarded as a lower bound of the makespan in the next step $m' = m - 1$. This may reduce the search space as constraints are tighter. However, this idea does not work effectively here since in each case where an optimal solution is found the makespan is actually determined by the earliest completion time $(r_j + p_j)$ of a test with a very long processing time. Normally during constraint propagation, a lower bound of the makespan is already constrained to be greater than or equal to the earliest completion times of all jobs. Therefore, it does not help CP when we try to include the same condition.

Table 2: Minimizing the makespan for the given number of prototypes.

Data	m	#Variables	#Constraints	Total Time (sec)	Makespan (day)
1	4	314	517	0.04	Infeasible
1	5	361	605	0.09	330[a]
1	14	784	1,397	0.18	330[a]
2	3	621	722	3.02	Infeasible
2	4	727	829	-[b]	-[b]
2	5	833	936	0.19	312[a]
2	34	3,907	4,039	1.14	312[a]
3	8	2,592	8,383	1,006.35	Infeasible
3	9	2,829	9,315	-[b]	-[b]
3	17	4,725	16,771	-[b]	-[b]
3	18	4,962	17,703	10.26[c]	381
3	19	5,199	18,635	13.45[c]	379
3	20	5,436	19,567	16.48	376[a]
3	77	18,945	72,691	91.87	376[a]
4	111	56,214	114,392	-[b]	-[b]
4	112	56,707	115,405	261.88[c]	517
4	113	57,200	116,418	319.57[c]	501
4	118	59,665	121,483	471.55[c]	481
4	123	62,130	126,548	832.21[c]	413
4	128	64,595	131,613	13,917.50	381[a]
4	162	81,357	166,055	21,430.75	381[a]

[a] Optimal solution
[b] No result within the time limit
[c] Computation time of the achieved solution

6 Conclusion

We have presented a scheduling problem from automobile industry where several hundreds of tests must be performed on vehicle prototypes. We have used the CP method to formulate the problem with the objective to minimize the makespan and applied it to solve real data sets. CP allows us to naturally express requirements like resource constraints and machine eligibility sets and then to efficiently solve the problem.

We are able to obtain good solutions for our scheduling problem within a reasonable computation time. Either the optimal or good feasible solutions are found even for the largest size problem which comprises about five hundreds tests and one hundred prototypes. The optimal makespan solution

can always be achieved when sufficient prototypes are available. However, it is more difficult to find an optimal solution if the number of machines is reduced. But in most cases, the solver manages to obtain good feasible solutions with the slight increase of makespan. If the number of given prototype is too low then we can prove that the problem becomes infeasible.

It is still quite hard and takes a very long time to exactly determine the minimum number of required prototypes. This is especially true for the data sets comprising several hundreds tests. In future, we are looking to determine better lower bound values. This may help a user to decide whether it is worth trying to further reduce the number of prototypes. Moreover, we will implement a more specific search strategy for this problem in order to improve the efficiency of constraint programming. In particular, it is interesting how we can use solutions obtained from solving previous iterations in finding an optimal or at least better feasible solution.

References

[1] P. Baptiste, C.L. Pape and W. Nuijten (2001), *Constraint-Based Scheduling: applying constraint programming to scheduling problems*, Kluwer.

[2] J.-H. Bartels and J. Zimmermann (2005), Scheduling Tests in Automotive R&D Projects, *Operations Research Proceedings 2005*, Springer, volume 11, 661 – 666.

[3] E. Danna and C.L. Pape (2004), *Constraint and Integer Programming: Toward a Unified Methodology*, chapter Two generic schemes for efficient and robust cooperative algorithms, 33 – 58, Kluwer.

[4] J. Hooker (2005), A Hybrid Method for the Planning and Scheduling, *Constraints* **10**(4), 385 – 401.

[5] ILOG S.A. (2003), *ILOG OPL Studio 3.7 Language Manual*.

[6] J. Lockledge, D. Mihailidis, J. Sidelko, and K. Chelst (2002), Prototype fleet optimization model, *Journal of the Operational Research Society* **53**, 833 – 841.

[7] M. Pinedo (2002), *Scheduling-Theory, Algorithm and Systems*, Prentice Hall, 2nd edition.

[8] R. Sadykov and L. Wolsey (2006), Integer programming and constraint programming in solving a multi-machine assignment scheduling problem with deadlines and release dates, *INFORMS Journal on Computing* **18**, 209 – 217.

[9] R. Scheffermann, U. Clausen, and A. Preusser (2005), Test-scheduling on vehicle prototypes in the automative industry, In *Proceedings of Sixth Asia-Pacific Industrial Engineering and Management Systems (APIEMS)*, volume 11, 1817 – 1830, Manila.

[10] A. Zakarian (2000), Performance analysis of product validation and test plans, Annual Progress Report, Center for Engineering Education and Practice, University of Michigan-Dearborn.

iSchedule, an Optimisation Toolkit for Complex Scheduling

Anne Liret
British Telecom, Tour Ariane - 5 place de la Pyramide - 92088 La Défense cedex, France,
anne.liret@bt.com

David Lesaint, Raphael Dorne, Chris Voudouris
British Telecom, BT Adastral Park - Marthlesham Heath - IP5RE Ipswich, United Kingdom,
{david.lesaint, Raphael.dorne, chris.voudouris}@bt.com

Most of real-world scheduling problems which include optimisation and feasibility aspects, are NP-hard. To tackle large scale optimization problems, meta-heuristic algorithms are often proposed to quickly obtain near-optimal solutions. However, they have difficulty to find feasible solutions on strongly constrained problems, whereas constraint programming can help as they efficiently handle feasibility problems. Moreover, despite their success, these methods often require complete re-implementation to solve new problems or variations of the same problem. To address these issues, the Framework approach of the intelligent Optimization Toolkit *iOpt* [5] provides three components for problem modelling, solving and visualizing, based on constraints and meta-heuristic algorithms. The problem representation is well-matched to heuristic search operations, thanks to the use of an invariants processing engine. As a consequence it reduces the time necessary to develop efficient solutions. Extensions of iOpt have been developed for specific domains. Such an extension is the *iSchedule* Toolkit which is dedicated to assist non-expert users in modelling scheduling applications.
This paper outlines iSchedule capabilities and illustrates their use on a particular scheduling problem, the mobile Workforce Scheduling problem, a Vehicle Routing-type problem with time windows, additional tasks and resource constraints, multiple concurrent objectives and subject to disturbances. This problem is of interest to BT in its field service operations.

Keywords: Heuristic Search, Real World Scheduling, Vehicle Routing, Invariant.

1. Introduction

Heuristic Search (HS) techniques are known for their efficiency and effectiveness in solving NP-Hard optimisation problems, though they need to be guided when the problem is strongly constrained and do not guarantee optimal solution. On the contrary constraint programming (CP) methods are known to efficiently solve feasibility problems and can help by removing unfeasible set of task allocations. Thus it is tempting to combine HS and CP to solve constrained optimisation problems. However, there is a need for a software toolkit which supports all the stages of designing and developing a system based on these techniques. For that reason, the *Intelligent Systems* lab in BT has developed the intelligent Optimisation toolkit *iOpt*, using object paradigm [5]. iOpt proved its approach on complex domains such as network design, automatic work allocation to field force, resource planning [9], timetabling, and transportation. A major extension of iOpt is the framework iSchedule, a toolkit for developing scheduling systems. iSchedule provides problem modelling and solving facilities based on constraints and heuristic searches.

Following contemporary thinking in software engineering, iOpt [2,3,5,6] allows code reuse to reduce development time for each new problem. Furthermore, application development is additive in the sense that the toolkit is enhanced by each new application, reducing further the effort in developing similar applications to those already included. iOpt is fully written in the Java programming language with all the acknowledged benefits associated with Java. iSchedule benefits from all the same features than iOpt, except they are specialised on scheduling. Especially, specific scheduling applications can be developed without having to review the core part of the system.

This paper outlines the main features of iSchedule (section 2) and describes its use on a particular scheduling problem, of interest to BT in its field service operations: the mobile Workforce Scheduling problem, a Vehicle Routing problem with time windows, additional tasks and resource constraints, multiple concurrent objectives, which is subject to disturbances (section 3). Then the paper concludes on the successful use of the iSchedule framework.

1.1 i*O*pt: the separation between problem model and algorithms

Based on a framework-oriented architecture, iOpt is designed to support rapid development of e-business applications that require optimisation functions (e.g. planning and scheduling). It is organised into three independent modules: a generic problem modelling framework (PMF) supported by an invariant library (IL), a heuristic search framework (HSF) and visualisation facilities for invariant networks, algorithm configuration and monitoring (Figure 1). This separation between problem model and the algorithm framework allows the user to run different algorithms on different problems at no cost. Further, several tools have been developed which enables the user to monitor the problem model or visualise heuristic search and construction search based algorithms [3]. For more details on iOpt please refer to [6].

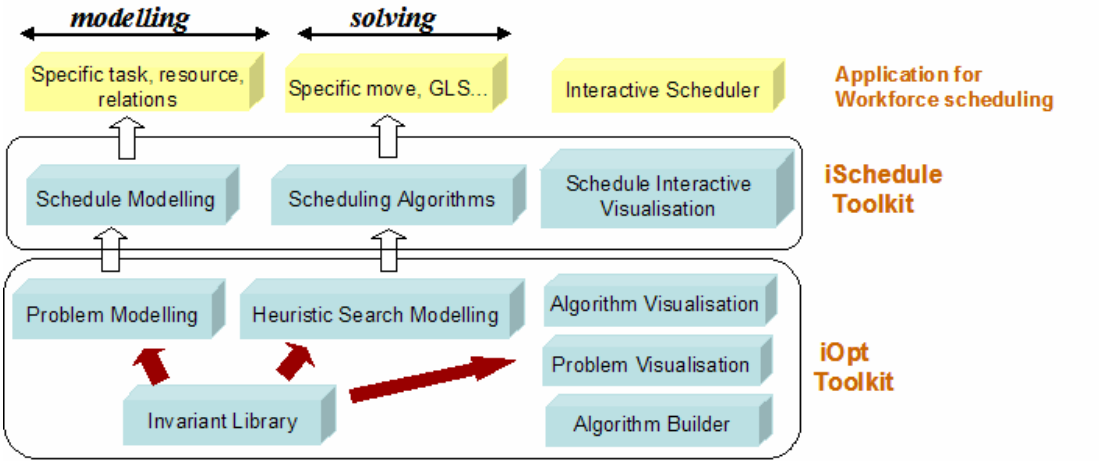


Fig.1. iOpt overview and how to specialise it

Figure 1 shows how to use iOpt/iSchedule to build a dedicated Workforce Scheduling solution. Only a very small part of iSchedule will be extended to meet the customer requirements thus reducing the development time while offering the high-performance algorithms based on iOpt technology.

3. iSchedule: a framework approach for building scheduling systems

iSchedule implements the basic concepts and business rules that are commonly encountered in scheduling problems with unary capacity resources. In addition, iSchedule manages interoperability between heterogeneous systems through XML exchange facilities: load/save a schedule, an algorithm, a scheduling problem specification.

3.1. Problem modelling

Main concepts are reified; a new application is addressed with iSchedule by reusing or overloading these concepts. The implementation can be done in a declarative and functional way thanks to the invariant underlying model.

The problem modelling framework is a high level framework based on an IL where the user can define scheduling problems with unary capacity resources as a Constrained Combinatorial Optimisation Problem (COP) with its decision variables representing resources and task times, a constraint network and an objective function, using the mathematical algebra available inside IL. The utilisation of IL, an originality of iOpt, supports problem modelling, constraint forward

checking, and search methods [7,8]. IL is very useful to quickly test and undo variable assignments [7,8]. Invariants are one-way constraints which, when grouped into a library, form a set of functions that provides a comprehensive mathematical algebra. It allows the definition of relations between variables, and performs incremental updates [1,4] when the value of one or more variables is changed. IL provides a number of built-in data types (Integer, Real, Boolean, String and Object) and operators which are available to the users to state their optimisation problem (figure 2). PMF incorporates solution management facilities by keeping the best solution, current solution or a population of solutions, which can be used by local search or population-based techniques. It can also be instructed to detect constraint violations and stop the evaluation algorithm at an early stage. Concretely, the developer does not have to be concerned with the details of the implementation and can focus his attention on what is specific to his application.

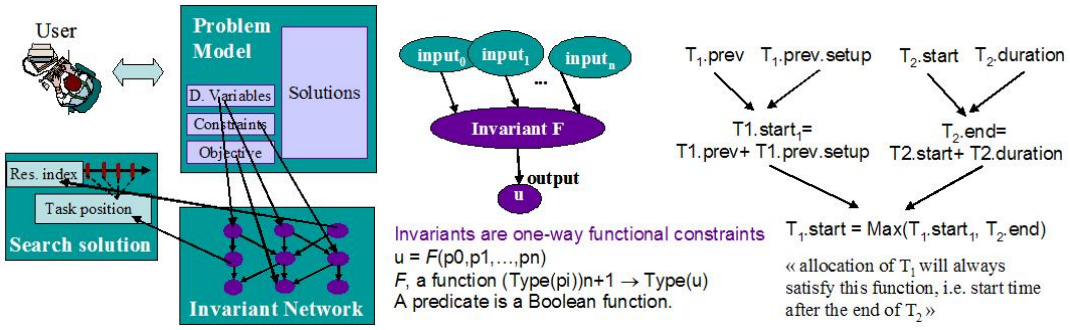


Fig.2. Problem statement using invariants.

Figure 2 gives an example of invariant-based precedence relation "T_1 starts after T_2 ends", and sketches how decision variables and heuristic search variables are directly connected to the invariant network. Indeed decision variables in the problem identify a job allocation, *i.e.* for each task, its resource, the previous activity and the next activity. Search variables identify the positions of a task in the sequences of job allocations which constitute scheduling solutions (see next section); they represent for each sequence the potential positions of a task and the resource index, whose value is mapped to the value of invariants.

The framework provides Java classes which hide the complexity of the decision model from the user, who focuses on problem modelling with entities from his application domain.

- **Activities and resources**: An activity represents any action that a resource can perform. The Scheduling Framework provides two different types of activities: customer activities called tasks and resource breaks. A task corresponds to an activity requested by a customer and that different resources may perform. A resource break is an activity that must be scheduled during the schedule horizon, e.g. lunch, meeting, home time. A resource represents any work unit that can undertake activities in sequence. In the case of Vehicle Routing, a resource is a pair (vehicle, driver) whose combined capacity, capability, and state determine which activities it can or cannot perform. The Scheduling Framework provides fixed and flexible breaks and implements the concept of *timeline f*or capturing time-dependent constraints on allocations.

- **Timelines:** Various resource attributes such as vehicle capacity affect the allocation of tasks to resources. Resource attributes may also evolve over time due to the execution of tasks. For instance, the load of a vehicle is constrained by its capacity hence preventing certain allocations to be made. The load is also modified whenever goods are delivered or picked-up at customer premises. Timelines simply capture the evolution of attribute values over time. The Scheduling Framework provides three types of timelines: capability timelines, capacity timelines, and state timelines.

- **Tasks:** A task is primarily defined by a duration, which may depend on the resource assigned. A task also features a start time and an end time which are connected by the usual relationship *start_time+ duration=end_time*, which is redefined in the case the option for splitting tasks over

breaks, is selected. The framework allows disabling tasks or resources, which will be then ignored in the further solution search runs.

- **Schedule:** A schedule is defined by a sequence of activities for each resource and a list of unallocated tasks together with global consistency and cost information on the schedule. The underlying computational model is the network of invariants representing the various constraints and costs that are built-in or user-defined.
- **Constraints**: The framework allows the statement of precedence constraints, resource constraint (same resource or not), time windows on start times or end times, resource compatibility with tasks.
- **Parallel relations**: They enforce several tasks to start at the same date simultaneously. A delay may be allowed which means there may be a maximum time gap between parallel tasks. In the case where resources are one-capacity, tasks are performed on different resources, which implies the checking of resource availability too.
- **Task costs:** They depend on the duration of the allocation, due dates, and the resource utilised for the allocation. For each task, a cost is incurred for not allocating the task in the schedule (unallocated cost), for letting a set-up period, for delaying the start of the task (lateness cost per unit of time), or for having a resource process the task (service time cost per unit of time).
- **Resource costs:** They rely on wages, overtime or travelling hours. For each resource, a cost is incurred for using the resource, for using the resource in overtime -whatever the task being processed (overtime cost), for using the resource per unit of time on any service work. The set-up time cost, for using the resource on any set-up period, can model a travelling cost too.
- **Service and set-up model**: They contain sub-models for the user to define their own cost, compatibility relation, or duration estimation of durations. They enable modelling resource setup/travel times and task duration. These are sub-models that can be easily linked to external systems, such as geographic information systems in the case of travel times, to offer realistic estimates. The cost sub-model is dedicated to contain as many terms as required to capture the particular business problem.

3.2 Scheduling Algorithms

Algorithms in iSchedule make use of heuristic search (through HSF), coupled to constraint satisfaction, but may be extended so as they integrate other types of algorithm, as graph theory, integer programming. Initially, the Heuristic Search Framework (HSF) was created to be a generic framework for the family of optimisation techniques known as Heuristic Search. It covers single solution methods such as Local Search, population-based methods such as Genetic Algorithms as well as hybrids combining one or more different algorithms. In HSF, the functionality of common HS algorithms is broken down into components (i.e. algorithmic parts) for which component categories are defined [2,5].

- **Search Component** represents the basic concept that could be encountered in an heuristic search, for example, a Neighbourhood in a Local Search, the Aspiration Criterion. A complete algorithm is a valid tree of search components (A valid tree is a tree of search components that can be executed).
- **Heuristic Solution** is the solution representation of an optimisation problem manipulated inside HSF, in the case of iSchedule, a set of sequences.
- **Heuristic Problem** is only an interface between an optimisation problem model implemented using PMF and HSF.

Per se, the Scheduling Framework does not make any scheduling decision. This is the role of external processes such as heuristic search algorithms. The framework simply provides the necessary interface to support schedule modifications: An insertion operation, which inserts a task before a specified activity in the schedule (tour construction); a swap operation, which swaps a task with a specified task in the schedule (tour improvement). These two operations suffice to

implement any complex move operator/neighbourhood. Of course, search algorithms require more than simple schedule modification operations, e.g. checking consistency, costing schedules, providing access to feasibility and cost information, saving the best solution encountered, resetting/swapping solutions, etc. These requirements are not specific to the scheduling domain but arise in all optimisation problems, which is why they are provided at the level of iOpt. For example, Figure 4 shows a scheduling-specific Fast Local Search algorithm where some algorithmic parts come from iOpt (red colour), other from iSchedule (green colour) and parts are newly created by the user (user-defined Task Selector for instance, in blue) to express a specific way to select the subset of tasks to be considered for insertion by the neighbourhood component.

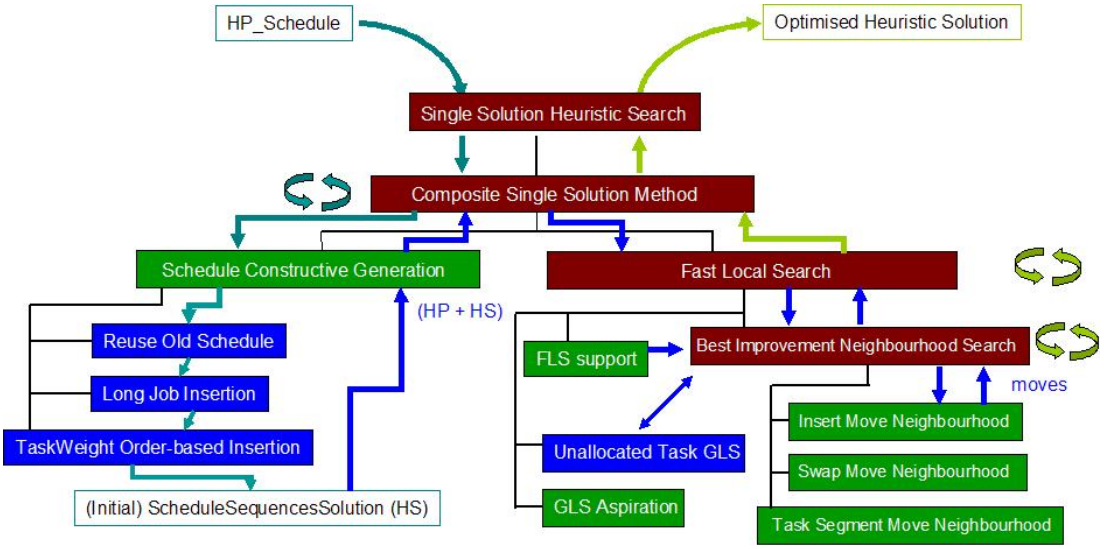


Fig.4. Example of a tree of algorithms parts for a Fast Local Search algorithm based on task insert, swap and segment exchange neighbourhoods.[1]

Among the parts readily available in iSchedule, we find the most famous components from local search operators such as segment neighbourhoods (2-opt, 2-opt*, OR-opt, Cross-exchange) and Critical Block based neighbourhoods, or initial schedule generation ("Longest Job First Generation", "Hardest Job First Generation", etc.) to population-based operators such as crossovers ("Uniform Resource Crossover", "Uniform Task Crossover" and "Route Crossover").

3. Application to real-world field force scheduling

iOpt and iSchedule have demonstrated their ability to provide quick prototyping of customised systems and solutions. Experience gained has been utilised to enhance the system, reducing further the effort in developing similar applications to those already included. Among these applications is the NGDS system (Next Generation Dynamic Scheduler), scheduling the routes of BT field engineers. With more than 150,000 tasks to be managed by 30,000 techs everyday, the BT workforce scheduling problem is a large scale problem [10]. Moreover the goal is quite challenging, covering service provision to business and residential customers, network maintenance, and fault repair, including the allocation of work, the execution and the tracking of jobs.

[1] FLS searches sub-neighborhoods of a given neighborhood using a given neighborhood search, deactivating those sub-neighborhoods which have no promising moves for future iterations or until they are re-activated as the result of a performed move. The search receives an input Heuristic Solution that is modified and returned as its current solution. A Schedule FLS support is a class describing the way the neighborhood search used by FLS works. A neighborhood search evaluates a set of moves provided by one neighborhood in order to select the next move, or moves to apply its current solution. BestImprovement goes through the list of moves provided by the neighborhood and performs the one that makes the best improvement to the solution.

The mobile workforce scheduling problem can be stated as a Vehicle Routing problem with time windows, additional tasks and resource constraints, and multiple concurrent objectives. The problem consists of mobile skilled resources, lunch breaks and breaks representing the night period (called home breaks), located tasks with time windows on start and / or end dates, long jobs which are likely to be split over breaks to be allocated, and parallel jobs. The objective of the problem is multi criteria: minimize the lateness and the number of unallocated tasks, minimize the travel, balance the resource usage so that at least each resource has at least one job per day, and minimize overtime. In short, it is about assigning the right job to the right resource at the right time to the right route with the right equipment.

The underlying computational problem is composed of a feasibility problem where the proposed allocations meet the constraint requirements so that no inconsistency occurs during execution, and an optimisation problem whose solutions agree with objectives. Multiple objectives become conflicting optimisation criteria due to schedule horizon restrictions, consequently leading to a problem where identifying a good quality solution (the greatest number of criteria satisfied) is time-consuming. To help handling the complexity of multiple objectives, NGDS explicitly makes objective terms available to the user, so as they can adjust cost expression, before the schedule search engine performs global optimisation. In addition NGDS complies with a variety of requirements that real-life scheduling domains must comply with: skill matching, routing, due dates, staff breaks, working time regulation constraints, task splitting on the fly. Built-in costs cover standard scheduling objectives and can be specialised to fit into certain requirements such as Quality of Service, travel and overtime penalties.

Typical outputs of NGDS include a Gantt Chart, table of task with information about their allocation, search progress view and statistics report (Figure 5). The tool offer a lightweight risk management facility as it helps making decision before task become failed business targets. Indeed it displays task allocation in different colours depending on task type, priority (importance score related to business target). Thus work managers can deduce task allocations that are candidate to jeopardy such as tasks about to fail (scheduled but too late or disruptions cause the task to be started or ended too late with respect to its due date or task not dispatched whereas its target is close to current time e.g. within the next hour). Also managers would be likely to distinguish an event that puts in jeopardy all the jobs in a tour from an event that just endangers the last job of the tour.

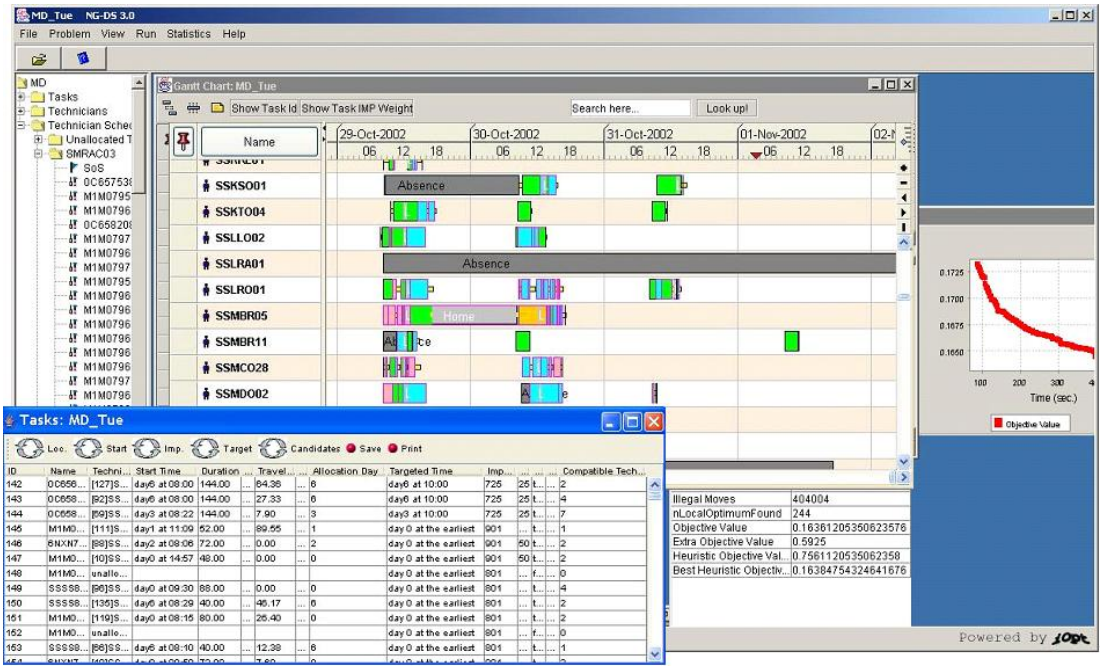


Fig.5. Typical outputs of NGDS. Task allocated in advance are displayed in blue, in green when scheduled on time of the expected day, in orange if there is up to one day late, and in pink tasks which would be allocated at the earliest because their target is already failed at the schedule start date.

Moreover the mobile workforce scheduling problem is subject to disturbances. In this particular context, to handle dynamic scheduling, a backend optimisation engine (predictive scheduling) is combined with online allocation capability (reactive scheduling). Estimated schedule is periodically (re)computed (rescheduling phase); the task allocation and input data are constantly changing while the schedule is executed during the day. To reduce the complexity of rescheduling we reuse past work, using the existing schedule as a starting point for the next scheduling. It suffices to identify and throw away invalid portions of the existing tours to build new tours. Also we extended iSchedule with capabilities which distinguish three sections in the schedule, the past, current and future schedule, depending on a current time of execution and an offset. The separation of the three schedules allows enforcing different consistency constraints on the schedule (for instance tasks which are near to be dispatched cannot be moved) and applying different solution search methods on each part.

Thank to iSchedule, the scheduling engine has been very quickly designed and developed (2 months with 2 persons). The remaining thing users would have to do was the algorithm and objectives fine-tuning (*via* graphical tool). We made intra-comparisons between results obtained with different algorithms configuration (figure 6).

4. Results

This section talks about results obtained when testing NGDS engine on a particular instance. The real-world daily region-localised problem deals with about 1000 jobs, up to 200 resources, and a scheduling horizon between 1 and 2 weeks. Jobs have start target dates (“startby” task), time windows (appointment) or completion target dates (“completeby” task).

Approaches used combine: 1) constructive search, that is first Schedule reuse, then parallel job insertion, long job insertion, then unallocated jobs insertion which can be random, task-importance ordered, technician tour-quality based. 2) Hill-Climber or Fast Local Search based on insert/swap/segment relocate moves, a GLS component to escape local optimum (some of the tasks are never visited due to a local good value found for travel cost), and best improvement neighbourhood search. It has been noticed that constructive search takes at least half of the running time (up to three minutes) and it is worthwhile treating long and parallel jobs beforehand as they are strongly constrained.

	FSLGLS_TO(alldays, day0)		FLSGLS_RO(alldays, day0)		HC_TO(alldays, day0)		HC_RO(alldays, day0)		FLS_TO(alldays, day0)		FLSGLS_TB(alldays, day0)		HC_TB		TO		TB		RO	
Scheduled tasks	707	343	697	379	705	374	697	379	710	368	707	346	706	343	704	374	704	312	696	373
Average travel per person (minutes)	60	38	63	43	70	49	67	47	61	43	78	54	81	51	68	45	95	54	63	43
Workstack per day per person	5	4	5	4	5	4	5	4	5	4	5	3	5	3	5	4	5	3	5	4
	HC=Hill-Climber			TO=Task ordering insertion			FLS= Fast Local Search without GLS						FSLGLS=FSL with lazy GLS							
	RO=Random ordering insertion						TB=Technician based insertion						HP=High priority							
	C- HP	C- MP	C- LP	AC0 HP	AC0 MP	AC0 LP	AC1 HP	AC1 MP	AC1 LP	AC2 HP	AC2 MP	AC2 LP	MP=Medium priority							
Assignable Tasks	37	39	119	66	78	13	49	96	1	53	170	11	LP=Low priority							
Scheduled Tasks	36	39	114	58	77	13	44	96	1	50	168	11	C- = Completby task failed at schedule start							
Total travel to Tasks	9.8	4.26	36.93	12.46	11.3	3.53	11.91	4.43	0.16	8.48	13.56	4.07	AC0 = task targeted to start on day 0 (today)							
'---'	'---'	'---'	'---'	'---'	'---'	'---'	'---'	'---'	'---'	'---'	'---'	'---'	AC1 = task targeted to start on day 1							
% of jobs scheduled	97	100	95	87	98	100	89	100	100	94	98	100	AC2 = task targeted to start on day 2 or later							

Fig.6. Table of statistics

The statistics table above shows number of scheduled tasks, average travel time per technician in minutes, and the number of jobs scheduled per day in average. These values are given on the right

column for the schedule in day 0 (first day of scheduling) and as an average over all days in the left column. For the run of FLSGLS_TO, the second table shows the percentage of scheduled jobs per category of jobs; where jobs are organised into categories depending on their due date and type. For a problem of 732 assignable jobs and 135 technicians, instance inspired from real workload of a Tuesday in June 2002. The run lasts 180 seconds. For this instance, FSLGLS evaluated 13870 moves in 85.35 seconds, in a network of 138927 invariants, and with a total of 248316 propagation sessions performed

All algorithms allocate a major part of the jobs, though FLS does better on that aspect, while keeping an affordable total travel cost. Hill-Climber ends up with a higher travel but it is partially due to the fact it accepts allocating tasks that are very far, though HC needs to have the travel cost weight higher than for the other algorithms, so as to maintain bounded travel. FLSGLS gives good compromised solution w.r.t. allocation and travel. Especially when looking at the statistics per task category (figure 6), we can see that NGDS achieves most of the time between 95% and 100% of scheduled tasks. Constructive search TB is focusing on the technician tour whereas TO inserts tasks according to the business importance (the highest before). RO performs task insertions where tasks are picked-up in a random order. The initial schedule computed by the TO constructive search is improved by the fast local search, keeping high priority (HP) tasks scheduled, as the percentages for HP tasks categories is high.

5. Conclusion

In society when the need of saving and rapid Return On Investment is increasing, object-oriented Framework approach looks promising to produce and maintain real-world scheduling systems. As an evidence, successful application of iSchedule and integration to client tools (mobility devices for job execution, resource planning, people management), as well as interoperability with web-based services. The iOpt / iSchedule package constitutes now a twofold offer in optimisation and scheduling: firstly a strong technical capability in modelling, solving and visualising complex problems inherited from iOpt, and secondly a high-level framework that allows rapid development of scheduling applications.

References

[1] B. Alpern, R. Hoover, B. Rosen, P. Sweeney and F. Zadeck (1990), Incremental evaluation of computational circuits, *ACM SIGACT-SIAM'89 Conference on Discrete Algorithms*, 32 – 42.

[2] R. Dorne and C. Voudouris (2001), HSF: A generic framework to easily design Meta-Heuristic methods, *4th Metaheuristics International Conference*, Porto, Portugal, 423 – 428.

[3] R. Dorne, C. Ladde and C. Voudouris (2003), Heuristic Search Builder: the iOpt' tool to visually build metaheuristic algorithms, *5th Metaheuristics International Conference*, Kyoto, Japan.

[4] S. Hudson (1991), Incremental Attribute Evaluation: A Flexible Algorithm for Lazy Update, *ACM Transactions on Programming Languages and Systems* **13**(3), 315 – 341.

[5] C. Voudouris, R. Dorne, D. Lesaint, A. Liret (2001), iOpt: A Software Toolkit for Heuristic Search Methods, *Principles and Practice of Constraint Programming - CP 2001*, ed. T. Walsh, Lecture Notes in Computer Science-Springer **2239**, 716 – 729.

[6] C. Voudouris and R. Dorne (2002), Integrating Heuristic search and One-Way Constraints in the iOpt Toolkit, *Optimization Software Class Libraries, Operations Research/Computer Science Interfaces* **18,** Chapter 6, Kluwer Academic Publishers, eds. S. Voss and D. Woodruff, 177 – 192.

[7] B. Zanden, R. Halterman, B. Myers, R. McDaniel, R. Miller, P. Szekely, D. Giuse and D. Kosbie (2001), Lessons Learned About One-Way, Dataflow Constraints in the Garnet and Amulet Graphical Toolkits, *ACM Transactions on Programming Languages and Systems (TOPLAS)* **23**(6) , 776 – 796.

[8] R. Dorne, P. Mills and C. Voudouris (2005), Solving Vehicle Routing using iOpt, *Proceedings of 5th Metaheuristic Intentional Conference*, Vienna, Austria.

[9] C. Voudouris, G. Owusu, R. Dorne, C. Ladde and B. Virginas (2003), ARMS. An Automated Resource Management System for British Telecommunications plc. *Proceedings of EURO/Informs Joint International Meeting*, Istanbul.
[10] D. Lesaint, C. Voudouris, N. Azarmi (2000), Dynamic Workforce Scheduling for British Telecommunications plc., *Interfaces, INFORMS.*

On a Generalized Graph Coloring/Batch Scheduling Problem

Giorgio Lucarelli[1], Ioannis Milis
Dept. of Informatics, Athens University of Economics and Business, 104 34, Athens, Greece, {gluc, milis}@aueb.gr
Vangelis Th. Paschos[2]
LAMSADE, Université Paris-Dauphine, 75016 Paris, France, paschos@lamsade.dauphine.fr

Abstract: We study a batch scheduling problem where jobs are no more independent but they are subject to (in)compatibility constraints described by an underlying graph. The problem is equivalent to a generalized weighted graph coloring problem. We study the frontier between polynomial and NP-variants of the problem as well as the approximability of NP-hard variants.

Keywords: Algorithmics, Batch Scheduling

1 Introduction

In several communication systems messages are to be transmitted in a single hop from senders to receivers through direct connections established by an underlying switching network. In such a system, a sender (resp. receiver) cannot send (resp. receive) more than one messages at a time, while the transmission of messages between different senders and receivers can take place simultaneously. The scheduler of such a system establishes successive configurations of the switching network, each one routing a non-conflicting subset of the messages from senders to receivers. Given the transmission time of each message, the transmission time of each configuration equals to the heaviest message transmitted. The aim of the scheduler is to find a sequence of configurations such that all the messages to be finally transmitted and the total transmission time to be minimized.

This problem is known as Time Slot Scheduling and it is equivalent to the parallel batch scheduling problem with (in)compatibilities between jobs, which is denoted as $1 \mid p-batch,\ graph \mid C_{max}$. Jobs are no more independent but they correspond to the edges of a weighted *graph*. Edge weights correspond to processing (transmission) times of jobs and the graph G describes (in)compatibilities between jobs: jobs corresponding to adjacent edges cannot be scheduled in the same batch (configuration).

The problem can be also formulated in graph-theoretic terms with senders and receivers corresponding to the vertices V of a graph $G(V, E)$, messages to its edges E, the lengths of messages to the weights of edges $w(e)$ and configurations to matchings. In this context, we ask for a partition $M = \{M_1, M_2, \ldots, M_s\}$ of the set of edges of the graph G into matchings, each one of weight $w(M_i) = \max\{w(e) | e \in M_i\}$, such that $w(M) = \sum_{i=1}^{s} w(M_i)$ is minimized. In the version we study in this paper preemptions are not allowed. In the following we shall refer to this problem as Batch Scheduling-Edge Coloring (BS-EC) problem.

Is is known that the BS-EC problem is strongly NP-hard and 7/6-inapproximable even for cubic planar bipartite graphs and edge weights restricted to be 1, 2 or 3 [6, 9]. On the other hand, a 2-approximation algorithm has been presented in [9] for bipartite graphs, which, in fact, applies also for general graphs. In addition, a $\frac{2\Delta-1}{3}$-approximation algorithm, for bipartite graphs

[1]This research project is co-financed by E.U.-European Social Fund (75%) and the Greek Ministry of Development-GSRT (25%).

[2]Part of this work has been carried out while the author was visiting the Department of Informatics of Athens University of Economics and Business.

of maximum degree Δ, has been presented in [5], which gives an approximation ratio of 5/3 for $\Delta = 3$. A new algorithm for bipartite graphs of $\Delta = 3$, presented in [6], achieves an approximation ratio equal to the 7/6-inapproximability result. Furthermore, an algorithm presented in [7] achieves approximation ratios less than two for bipartite graphs with $4 \leq \Delta \leq 7$. The preemptive variant of the BS-EC problem on bipartite graphs has been also studied [4, 1, 3].

Moreover, a large body of work is concentrated on the analogous Batch Scheduling-Vertex Coloring (BS-VC) [2, 8, 5, 6, 7, 10, 11]. In this problem jobs correspond to the weighted vertices of a given graph G and adjacent jobs (vertices) cannot be scheduled in the same batch. In graph theoretic terms we ask for a partition of the vertices of G into independent sets, each one of weight equal to the maximum weight of its vertices, so that the total weight of the partition is minimized. It is clear that the BS-EC problem on a general graph G is equivalent to the BS-VC problem on the line graph, $L(G)$, of G and thus any algorithm for the BS-VC problem applies also to the BS-EC problem. However, this is not true for special graph classes, since the line graph of a special graph (e.g. bipartite or tree) is not anymore in the same special class.

In this paper we study the BS-EC problem. In the next section we give the notation we use and some preliminaries. As the complexity of the problem on trees remains open, in Section 3, we present a polynomial algorithm for trees of bounded degree and a polynomial algorithm for stars of chains. Finally in Section 4 we present an approximation algorithm for bipartite graphs which beats the known ones for bipartite graphs of maximum degree $\Delta \leq 12$.

2 Preliminaries

In the following we consider the non-preemptive BS-EC problem on an incompatibility weighted graph $G = (V, E)$. By $d(v),\ v \in V$, we denote the degree of vertex v and by $\Delta(G)$ (or simply Δ) the maximum vertex degree of G. We define the degree of each edge $e(u, v) \in E$ as $d(u, v) = d(u) + d(v)$ and $\Delta'(G)$ (or simply Δ') denotes the maximum edge degree. Moreover, we consider the edges of G sorted in non-increasing order of their weights, $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_m)$. Thus, e_1 denotes the heaviest edge of G.

By OPT we denote the cost of an optimal solution to the BS-EC problem. We also assume that in such an optimal solution the graph is decomposed into s^* matchings each one of weight w_i^*. Without loss of generality we consider the matchings of any solution in non-increasing order of their weights, i.e. $w(M_1) \geq w(M_2) \geq \ldots \geq w(M_s)$, and for the optimal solution $w_1^* \geq w_2^* \geq \ldots \geq w_{s^*}^*$.

Proposition 1 *For the number of matchings, s^*, in an optimal solution of the* BS-EC *problem on a graph G, it holds that $\Delta \leq s^* \leq \Delta' - 1 \leq 2\Delta - 1$.*

Proof: Obviously, any solution consists of at least Δ matchings. Assume that an optimal solution consists of Δ' or more matchings. As any edge of G has at most $\Delta' - 2$ neighbor edges, it follows that for each edge e in any $(\Delta' + i)$-th matching, $i > 0$, there is one of the first $\Delta' - 1$ matchings where e can be moved without increasing the weight of this matching (recall that matchings are considered in non-increasing order of their weights). ■

In the case where all edges have the same length the BS-EC problem is equivalent to the classic *edge coloring* problem, where the objective is to minimize the number of colors (matchings) required in order to assign different colors to neighbor edges. It is well known that this problem is NP-hard in general graphs, but it is solvable in polynomial time in bipartite graphs, using as many colors as the maximum degree of the graph. Applying such an algorithm for a weighted bipartite graph we obtain a Δ-matchings solution, in general non optimal, to the BS-EC problem.

The BS-EC problem is also polynomial for graphs of maximum degree $\Delta = 2$. This result follows from the same variant of the BS-VC problem. In [7] has been presented an $O(|V|^2)$ algorithm for the BS-VC problem on chains, which can be easily adapted for the BS-VC problem on graphs of maximum degree $\Delta = 2$ (collections of chains and cycles). On the other hand, if G is a graph of maximum degree two then its line graph $L(G)$ is also a graph with $\Delta(L(G)) = 2$.

Theorem 1 *An optimal solution to the* BS-EC *problem for graphs of maximum degree* $\Delta = 2$ *consists of at most three (i.e., two or three) matchings and can be found in* $O(|E|^2)$ *time.*

3 Trees

As the complexity of the BS-EC problem on trees remains open, in this section we first present a polynomial algorithm for a related decision problem called Feasible k-Coloring. This algorithm is then used to derive a polynomial algorithm for the BS-EC problem on trees of bounded degree. We also present a polynomial algorithm for stars of chains.

3.1 Feasible k-Coloring and bounded degree trees

The Feasible k-Coloring problem is formally defined as following. An analogous problem is also defined and solved in [11] for the BS-VC problem on trees.

Feasible k-Coloring:
Instance: A tree $T(V, E)$, a weight function $w(e) : E \to \mathbb{N}$ and a sequence of k integer weights $a_1, a_2, \ldots, a_k$, such that $a_1 \geq a_2 \geq \ldots \geq a_k$.
Question: Is there a partition of the edges E of T into exactly k matchings $M_1, M_2, \ldots, M_k$, such that $w(M_j) \leq a_j,\ 1 \leq j \leq k$?

We consider the tree T rooted at an arbitrary vertex, r. For each edge $e = (v, u)$ we define u to be the most distant from r endpoint of e, and $T(e)$ to be the subtree of T rooted at u. We denote by $S(e) \subseteq \{M_1, M_2, \ldots, M_k\}$ to be the set of matchings in which edge e can belong in order the subtree $T(e) \cup \{e\}$ to be feasibly colorable.

Our algorithm initializes the sets $S(e)$ for each leaf edge e to contain the matchings that are heaviest than its weight $w(e)$. Moreover, a fictive edge e_0 of weight $w(e_0) = 0$ is connected to the root of the tree, as in Figure 1.a, in order to treat the root of the tree as the rest vertices. The Feasible k-Coloring algorithm follows.

Algorithm 1
1. Initialization:
   Leaf edges: $S(e) = \{M_j | 1 \leq j \leq k$ and $w(e) \leq a_j\}$
   Rest edges: $S(e) = \{\}$
   Add a fictive vertex r', a fictive edge $e_0 = (r', r)$ with $w(e_0) = 0$
   and a fictive matching M_0 with $w(M_0) = a_0 = 0$
2. For each $e \in E \cup \{e_0\}$ in post-order do
3. For each matching M_j such that $w(e) \leq a_j$ do
4. If there is coloring of $T(e) \cup \{e\}$ such that $e \in M_j$ then $S(e) = S(e) \cup \{M_j\}$
5. If $S(e) = \emptyset$ then return
6. Create a feasible coloring using the $S(e)$'s

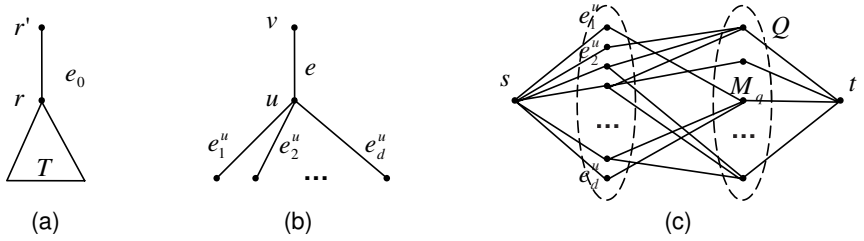


Figure 1: (a) The fictive edge e_0. (b) An instance of the graph in line 4 of Algorithm 1. (c) The flow network constructed in line 4 of Algorithm 1. All edges have weight equal to 1.

In line 4, Algorithm 1 decides if a feasible coloring for the subtree $T(e) \cup \{e\}$ exists (see Figure 1.b). For each edge $e = (v, u)$, we define $E_u = \{e_1^u, e_2^u, \ldots, e_d^u\}$ to be the set of edges from u to its children (recall that u is the most distant from the root of the tree endpoint of e). Each edge e_i^u can belong in one of the matchings in its $S(e_i^u)$. Let Q be the union of the sets $S(e_i^u)$, but the matching M_j edge e is assigned to, i.e. $Q = \bigcup_{i=1}^{d} S(e_i^u) - \{M_j\}$. We create a bipartite graph $B(E_u, Q; S)$, where there is an edge between $e_i^u \in E_u$ and $M_q \in Q$ iff $M_q \in S(e_i^u)$. Then we create a flow network F by joining a source vertex s to each vertex in E_u and a terminal vertex t to each vertex in Q, as in Figure 1.c. We assign to all the edges in F a weight equal to 1. Then, it follows that there is coloring of $T(e) \cup \{e\}$ such that $e \in M_j$ iff there is in F an $s - t$ flow of value d.

In line 6, Algorithm 1 creates a partition of the edges of T into matchings, by assigning each edge e of T, in pre-order, to an arbitrary matching in its set $S(e)$.

Algorithm 1 performs $k \cdot |E|$ iterations and in each one of them runs a maximum flow algorithm of polynomial complexity (e.g. Goldberg and Tarjan's algorithm of complexity $O((|V| + k)^3)$). Thus, the next Theorem follows.

Theorem 2 *There a polynomial time algorithm for the Feasible k-Coloring problem.*

Algorithm 1 can be used to solve the BS-EC problem on trees, as following.

Algorithm 2

```
1. For k = Δ to Δ' − 1 do
2.   For all (|E| choose k) combinations of edge weights run Algorithm 1
3. Return the best of the solutions found
```

Line 2 of Algorithm 2 is repeated $O(\Delta \cdot |E|^{\Delta})$ times, and therefore it is polynomial only for trees of polynomially bounded degree.

3.2 Stars of chains

A star of chains consists of p chains $C_1, C_2, \ldots, C_p$ all starting from a common vertex, say u. We consider each chain C_i, $1 \leq i \leq p$, starting from u with an edge e_i^u which we call start edge. We assume also that $w(e_1^u) \geq w(e_2^u) \geq \ldots \geq w(e_p^u)$.

Lemma 1 *For an optimal solution of the* BS-EC *problem on a star of chains the following hold:*
i) The number of matchings s^* equals to either p or $p + 1$.
ii) Only $k \leq 3$ matchings have cardinality $|M_j| > 1$.
iii) At least the $k - 1$ heaviest start edges appear in these k matchings.

Proof:
i) By Proposition 1, $\Delta \leq s^* \leq \Delta' - 1$. Here, $\Delta = p$ and $\Delta' = p + 2$.
ii) Assume that an optimal solution has more than three matchings of cardinality $|M_j| > 1$. Consider those matchings sorted in non-increasing order of their weights. Each non start edge e has at most 2 neighbor edges. So, such an edge e can be moved in one of the three first heaviest matchings.
iii) Consider first that $k = 2$. Assume that in the optimal solution the heaviest start edge e_1^u does not belong to neither of two matchings of cardinality $|M_j| > 1$. Then e_1^u can be either inserted in one of those two matchings (if this does not contain another start edge) or e_1^u can replace an existing start edge. In both cases the cost of the optimal solution decreases or remains the same. Assume next that $k = 3$. As in the previous case e_1^u can be inserted in one of the three matchings of cardinality $|M_j| > 1$ and e_2^u can be inserted in one of the rest two of those matchings. ■

In the following we distinguish between two cases according to possible number of matchings in an optimal solution, i.e. $p + 1$ or p.

If an optimal solution consists of $p+1$ matchings then it contains exactly one matching without any start edge. Algorithm 3 finds such an optimal schedule with $p + 1$ matchings.

Algorithm 3
1. Remove from the star the $p-2$ lightest start edges
   (this creates a graph H consisting of $p-1$ chains)
2. Find an optimal solution $S^*(H)$ for the graph H, using Theorem 1
3. If there are 3 non empty matchings in $S^*(H)$ then
4. Return the solution consisting of these 3 matchings of $S^*(H)$ plus $p-2$ matchings each one containing one of the removed $p-2$ lightest start edges

Note that Algorithm 3 is possible to return $p - 1$ matchings of $|M_i| = 1$, in the case where the one of the three matchings of $S^*(H)$ found in line 2 consists of a single edge. Taking into account Lemma 1, it follows that Algorithm 3 returns the optimal solution of $p + 1$ matchings since: (i) the $p - (k - 1)$ matchings of cardinality $|M_i| = 1$ contain the $p - (k - 1)$ lightest start edges (one per each matching) and (ii) the cost of k matchings is optimal. The complexity of Algorithm 3 is dominated by line 2 and by Theorem 1 it is $O(|E|^2)$.

If an optimal solution consists of p matchings, then each of them contains a start edge. Algorithm 4 returns such an optimal schedule with p matchings.

Algorithm 4
1. For $i = 3$ to p do
2. Remove $p-3$ start edges $e_3^u, e_4^u, \ldots, e_{i-1}^u, e_{i+1}^u, \ldots, e_p^u$
   (this creates a star T of 3 chains and a graph H of $p-3$ chains)
3. Find the optimal solution $S^*(T)$ using Algorithm 2
4. If there are exactly 3 matchings in $S^*(T)$ then
5. Find the optimal solution $S^*(H)$ using Theorem 1
6. Combine the solutions $S^*(T)$ and $S^*(H)$ into exactly 3 matchings
7. Find a solution for the initial star consisting of these 3 matchings plus $p-3$ matchings each one containing one of the removed $p-3$ start edges
8. Return the best solution found

Algorithm 2 is used in line 3 since T is a bounded degree tree with $\Delta = 3$ and it returns an optimal solution $S^*(T)$ of at least three matchings. In line 6, a 3-matchings optimal solution for the

edges in T and H can be obtained by considering the matchings in both solutions in non-increasing order of their weights and merging the matchings of each solution having the same rank. The optimality of the solution that Algorithm 4 returns follows by Lemma 1 using the same arguments as for Algorithm 3. The complexity of the algorithm is dominated by line 3 which takes $O(|E|^3)$ time and is executed $\Delta - 2$ times.

Theorem 3 *The* BS-EC *problem on stars of chains can be solved optimally in* $O(\Delta \cdot |E|^3)$ *time.*

4 Bipartite graphs

In this section we present an algorithm which improves the best known approximation ratios for the BS-EC problem in bipartite graphs of maximum degree $4 \leq \Delta \leq 12$.

Our algorithm generalizes the idea behind the 7/6-approximation algorithm for bipartite graphs of $\Delta = 3$, proposed in [6], to general bipartite graphs. In fact, our algorithm splits repeatedly a given bipartite graph G, of maximum degree Δ, into three edge induced subgraphs. To describe this partition as well as our algorithms let us introduce some additional notation. Recall that we consider the edges of G sorted in non-increasing order of their weights i.e., $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_m)$. For this order of edges we denote by $G_{j,k}$, $j \leq k$, the subgraph of G induced by the edges $e_j, e_{j+1}, \ldots, e_k$. By $\Delta_{j,k}$ we denote the maximum degree of graph $G_{j,k}$. By convention, we define $G_{m+1,m}$ as an empty graph. By j_q we denote the maximum index such that $\Delta_{1,j_q} = q$. It is clear that $j_1 < j_2 < \ldots < j_\Delta = m$.

In general our algorithm examines, for each pair of indices j, k, $j = 1, 2, \ldots, j_{\Delta-1}$, $k = j + 1, \ldots, m$, a partition of graph G into three edge induced subgraphs: the graph $G_{1,j}$, induced by the j heaviest edges of G, the graph $G_{j+1,k}$, induced by the next $k-j$ edges of G, and the graph $G_{k+1,m}$, induced by the $m-k$ lightest edges of G. For each one of these partitions the algorithm checks the existence of two different specific matchings in graph $G_{j+1,k}$. For each one of these matchings ,if exists, a solution for the BS-EC problem on graph G is obtained. Finally, the algorithm returns the best among all the solutions found.

Algorithm 5 (G)

```
1.  For j = 1, 2, ..., j_{Δ-1} do
2.    For k = j + 1 to m do
3.      If there is a matching M in G_{j+1,k} such that each vertex in G_{1,k}
              of degree Δ is saturated
4.        Create a solution for G_{1,k} by concatenating a solution of Δ − 1 matchings
              for G_{1,k} − M and the matching M
5.        Complete greedily this solution with the edges of G_{k+1,m}
6.      If j ≤ j_2 then find an optimal solution S_{1,j} for G_{1,j} using Theorem 1
7.      else find a solution S_{1,j} for G_{1,j} by calling Algorithm 5 (G_{1,j})
8.      If there is a matching M' in G_{j+1,k} such that each vertex in G_{j+1,k}
              of degree Δ is saturated and M' fits in S_{1,j}
9.        Create a solution S_{j+1,k} with Δ − 1 matchings for G_{j+1,k} − M'
10.       Concatenate S_{1,j} and S_{j+1,k} and complete greedily this solution
              with the edges of G_{k+1,m}
11. Return the best solution found in Steps 5 and 10
```

In Lines 5 and 10 the algorithm completes a partial solution by examining the remaining lightest edges one by one and assigning them to the first matching where they fit. If such a matching does

not exist then a new one is created. As both partial solutions consist of at most $2\Delta - 1$ matchings, the complete solutions obtained will consist also of at most $2\Delta - 1$ matchings, by the arguments in the proof of Proposition 1.

The following lemma bounds the weight W of the solution obtained by Algorithm 5. By ϱ_q we define the approximation ratio of our algorithm for a graph with maximum degree q. By definition, $\varrho_1 = \varrho_2 = 1$, since for graphs of maximum degree 1 or 2 the BS-EC problem can be solved in polynomial time. Recall that, w.l.o.g., we consider the matchings of an optimal solution in non-increasing order of their weights, i.e., $w_1^* \geq w_2^* \geq \ldots \geq w_{s^*}^*$.

Lemma 2 *Algorithm 5 returns a solution of cost:*

$$W \leq \min \left\{ (\Delta - 1) \cdot w_1^* + w_\Delta^* + (\Delta - 1) \cdot w_{\Delta+1}^*, \min_{1 \leq q \leq \Delta - 1} \{ \varrho_q \cdot \sum_{i=1}^{q} w_i^* + (\Delta - 1) \cdot w_{q+1}^* + (\Delta - q) \cdot w_{\Delta+q}^* \} \right\}$$

Proof:
For the first term in the lemma's inequality consider the solution obtained by the Lines 3–5 of the algorithm in the iteration (j, k) where $w(e_{j+1}) = w_\Delta^*$ and $w(e_{k+1}) = w_{\Delta+1}^*$ (note that if $k = m$ the algorithm examines the case where $w_{\Delta+1}^* = 0$). In this iteration the matching M exists, since in the optimal solution the edges of $G_{1,k}$ belong in at most Δ matchings and the Δ-th matching contains edges of weight at most $w(e_{j+1})$. The weight of M is at most w_Δ^*, while the weight of the solution found for $G_{1,k} - M$ is bounded by $(\Delta - 1) \cdot w_1^*$. The greedy step in Line 5 creates at most $\Delta - 1$ matchings, each one of weight at most $w_{\Delta+1}^*$. Therefore, $W \leq (\Delta - 1) \cdot w_1^* + w_\Delta^* + (\Delta - 1) \cdot w_{\Delta+1}^*$.

For the second term in the lemma's inequality, consider the solutions obtained by the Lines 6–10 of the algorithm for $\Delta - 1$ different iterations (i, k). For $j \leq j_q$, consider the iteration k where $w(e_{j+1}) = w_{q+1}^*$ and $w(e_{k+1}) = w_{\Delta+q}^*$. In this iteration, the matching M' exists, since in the optimal solution the edges of $G_{j+1,k}$ belong in at most $\Delta - 1$ matchings. If $q = 1$ or 2, the algorithm creates an optimal solution for $G_{1,j}$ using Theorem 1, since $\Delta_{1,j} \leq 2$. If $q \geq 3$, the algorithm creates an approximation solution for $G_{1,j}$. In both cases, the edges in $G_{1,j}$ are a subset of the edges of the q heaviest matchings in the optimal solution. Thus, it holds that $S_{1,j} \leq \varrho_q \cdot \sum_{i=1}^{q} w_i^*$, where $\varrho_1 = \varrho_2 = 2$. The weight of the solution found for $G_{j+1,k} - M'$ is bounded by $(\Delta - 1) \cdot w_{q+1}^*$, since $\Delta(G_{j+1,k} - M') \leq \Delta - 1$ and for each edge $e \in G_{j+1,k} - M'$ it holds that $w(e) \leq w_{q+1}^*$. The greedy step in Line 10 creates at most $\Delta - q$ matchings, each one of weight at most $w_{\Delta+q}^*$. Summing up the costs of these three partial solutions the lemma follows. ■

The Algorithm 5 performs $O(|E|^2)$ iterations. The recursion in Line 7 is of depth 1, since for each partition j', k' of the graph $G_{1,j}$ the solution for the graph $G_{1,j'}$ is already computed in some previous iteration. The matchings M and M' can be computed in polynomial time by generalizing the ideas proposed in [6] for bipartite graphs of $\Delta = 3$.

For $\Delta = 4$, Algorithm 5 returns a solution of cost W, for which holds that:

$$\begin{aligned} W &\leq 3 \cdot w_1^* + w_4^* + 3 \cdot w_5^* \\ W &\leq w_1^* + 3 \cdot w_2^* + 3 \cdot w_5^* \\ W &\leq w_1^* + w_2^* + 3 \cdot w_3^* + 2 \cdot w_6^* \\ W &\leq 7/6 \cdot (w_1^* + w_2^* + w_3^*) + 3 \cdot w_4^* + w_7^* \end{aligned}$$

Multiplying these inequalities by 44/458, 66/458, 99/458 and 138/458, respectively, and adding them we obtain a ratio $\varrho_4 = 458/347 \simeq 1.32$ for the BS-EC problem in graphs with maximum degree $\Delta = 4$. In a same way, we can compute the ratio ϱ_Δ for different values of Δ.

The following table summarizes the approximation ratio achieved by Algorithm 5 as well as the previous best known ratio for $\Delta = 3, 4, \ldots, 12$.

Δ	3	4	5	6	7	8	9	10	11	12
previous ratio	$1.17^{[6]}$	$1.61^{[7]}$	$1.75^{[7]}$	$1.86^{[7]}$	$1.95^{[7]}$	$2^{[9]}$	$2^{[9]}$	$2^{[9]}$	$2^{[9]}$	$2^{[9]}$
our ratio	1.17	1.32	1.45	1.56	1.65	1.74	1.81	1.87	1.93	1.98

Algorithm 5 gives better results than the approximation ratio $(2\Delta - 1)/3$ given in [5], for any Δ, as well as than the 2-approximation algorithm given in [9], for bipartite graphs of $\Delta \leq 12$. Furthermore, for bipartite graphs of $4 \leq \Delta \leq 7$ our algorithm improves the best known results given in [7]. For bipartite graphs of maximum degree $\Delta = 3$, Algorithm 5 reduces to the 7/6-approximation algorithm proposed in [6].

References

[1] F. Afrati, T. Aslanidis, E. Bampis, I. Milis (2005), Scheduling in Switching Networks with Set-Up Delays, *Journal of Combinatorial Optimization* **1**, 49 – 57.

[2] P. Brucker, A. Gladky, H. Hoogeveen, M. Koyalyov, C. Potts, T. Tautenham, S. van de Velde (1998), Scheduling a Batching Machine, *Journal of Scheduling* **1**, 31 – 54.

[3] J. Cohen, E. Jeannot, N. Padoy, F. Wagner (2006), Messages Scheduling for Parallel Data Redistribution between Clusters, *IEEE Trans. on Parallel and Distributed Systems* **17**, 1163 – 1175.

[4] P. Crescenzi, X. Deng, Ch. H. Papadimitriou (2001), On Approximating a Scheduling Problem, *Journal of Combinatorial Optimization* **3**, 287 – 297.

[5] M. Demange, D. de Werra, J. Monnot, V. Th. Paschos (2002), Weighted Node Coloring: When Stable Sets Are Expensive, In: *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 114 – 125.

[6] D. de Werra, M. Demange, B. Escoffier, J. Monnot, V. Th. Paschos (2004), Weighted Coloring on Planar, Bipartite and Split Graphs: Complexity and Improved Approximation, In: *International Symposium on Algorithms and Computation (ISAAC)*, 896 – 907.

[7] B. Escoffier, J. Monnot, V. Th. Paschos (2006), Weighted Coloring: further complexity and approximability results, *Information Processing Letters* **97**, 98 – 103.

[8] G. Finke, V. Jost, M. Queyranne, A. Sebő (2004), Batch processing with interval graph compatibilities between tasks, *Cahiers du laboratoire Leibniz*; available at http://www-leibniz.imag.fr/NEWLEIBNIZ/LesCahiers/index.xhtml.

[9] A. Kesselman, K. Kogan (2004), Non-preemptive scheduling of optical switches, In: *IEEE Global Telecommunications Conference (GLOBECOM)* **3**, 1840 – 1844.

[10] S. V. Pemmaraju, R. Raman, K. R. Varadarajan (2004), Buffer minimization using max-coloring, In: *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 562 – 571.

[11] S. V. Pemmaraju, R. Raman (2005), Approximation Algorithms for the Max-coloring Problem, In: *International Colloquium on Automata, Languages and Programming (ICALP)*, 1064 – 1075.

Inventory Routing Problem Solved by Heuristic Based on Column Generation

Sophie Michel François Vanderbeck
University Bordeaux 1, 351 Cours de la Libération, 33405 Talence Cedex, FRANCE,
{smichel, fv}@math.u-bordeaux1.fr

We consider an application of the inventory routing problem. A fleet of vehicles is devoted to collecting a single product from geographically dispersed sites. Each site has its own accumulation rate and stock capacity. On each visit, the vehicle empties the stock. At the tactical level, the objective is to minimize the fleet size and an estimate of the distance travelled. Moreover, for practical purposes, routes must be geographically clustered and the planning must be repeated over the time horizon with constrained periodicity. We develop a truncated branch-and-price-and-cut algorithm combined with rounding and local search heuristics that yield both primal solutions and dual bounds. Periodic plannings are generated for vehicles by solving a multiple choice knapsack problem. The issues related to the construction of the customer plannings are dealt with in a master program. The key to the success of the approach is the use of a state-space relaxation technique in formulating the master program to avoid the symmetry in time. Real-life instances of the problem are solved with reasonable optimality gaps.

Keywords: Real World Application, Vehicle Routing, Planning, Primal Heuristics.

Introduction

The Inventory Routing Problem (IRP) combines issue of vehicle routing for pick-up or deliveries with inventory management at customer sites. Three decisions have to be made: (i) when to serve a customer; (ii) how much to deliver to a customer when it is served; and (iii) which delivery routes to use. Many variants are discussed in the literature [1, 2, 5]. The existing approaches tend to make restrictive assumption (such as assuming a fixed partition policy) or to adopt a hierarchical optimization scheme where planning is decided before routing. Most approaches are heuristics with no warranty on the deviation to optimality and are specific to the problem variant.

We consider real-life instances of the problem (using data coming from our industrial partner). In the application that motivates our study, the stock management policy is simple with deterministic consumption rate and an order-up-to-level policy. Section 1 describes this problem and our assumptions. The problem is well-suited for decomposition, hence our approach relies on Dantzig-Wolfe reformulation [10] as outlined in Section 2. The latter eliminates the symmetry in vehicle indexing (vehicles are identical) but still suffers from a symmetry in time (shifting the starting time of routes in a periodic solutions can define a symmetric solution). Hence, we model average behavior by considering a single aggregate variable measuring how many times a specific route and associated delivery pattern is used over all possible starting dates. Cutting planes are added to the master to improve the formulation. Dual bounds are obtained by LP relaxation and tightened through partial branching. Heuristics based on column generation give primal bounds. Sections 3 and 4 present methodology to obtain respectively dual and primal bounds as well as computational results. Primal heuristics developed here in a column generation context are generic and can be used for others problems.

1 The Problem

The application considered here concerned the design of routes for collecting a single product from customers who accumulate it in their stock. At the tactical planning level, filling rates at collection points are seen as determinist. The stock management rule is simple: at each pick-up, the stock is emptied (this is the equivalent of an "order up to level" policy). Thus, the collected quantities can be normalized in number of periods that have passed since the last visit. The customer stock capacity implies a maximum interval between two visits, $tmax$. The stock management costs reduce to the transportation cost.

In search for a periodic solution, we restrict the solution space by imposing that route periodicity are selected from a restricted set P: for example in our tests $P = \{1, 2, 3, 4, 5, 6\}$. For each route, we must select its periodicity $p \in P$ and its first occurrence, i.e., its starting date $s \leq p$. Then, the solution is H periodic where H is bounded by the least common multiple of the periodicities (in our example $H \leq T = 60$). T is the maximal length of the regeneration cycle. The planning requirements boil down to ensuring that the stocks produced on each period of the regeneration cycle are picked-up by some vehicle route.

The exact routing of vehicles is considered as an operational issue. At the tactical planning level, we define a route by the cluster of visited customer sites and the specific quantities that are collected on each visited point (their sum does not exceed the vehicle capacity). The operational routing cost is approximated by the sum of the distances to the cluster center defined as one of the visited points (it is the seed of the route). Thus, each route is associated to a star in the graph of customer points. This measure favor the grouping of customers that are geographically close to each other. The planning constraints will induces the formation of clusters that group customers sharing the same frequency of collect.

The cost function includes fixed costs per vehicle (the main objective being to minimize the number of vehicle used), but it also includes our cluster approximation of routing costs that yields clusters of points that are geographically gathered around their seed.

2 A Dantzig-Wolfe Decomposition approach

The problem decomposes into planning issues on one hand and routing issues on the other. We formulate the planning problem in terms of variables associated with the selection of routes. The definition of a route specifies the visited customers, the quantities picked-up at each site (expressed in number of periods worth), the periodicity and the starting date.

Once the periodicity, p, of a route is fixed, as well as its starting date, s, and its seed, k, the problem of selecting the members of the cluster and associated picked-up quantities reduces to a variant of the multiple choice knapsack problem: let $\phi_{i\ell}$ equal to 1 if the customer i is in the cluster and the quantity that is collected is the production of ℓ periods; the associated profit is $p_{i\ell}$; the

knapsack formulation is

$$\max \sum_{i\ell} p_{i\ell}\phi_{i\ell} \tag{1}$$

$$\sum_{\ell} \phi_{k\ell} = 1 \tag{2}$$

$$\sum_{\ell} \phi_{i\ell} \leq 1 \qquad \forall\, i \neq k \tag{3}$$

$$\sum_{i\ell} \ell d_i \phi_{i\ell} \leq W \tag{4}$$

$$\phi_{i\ell} \in \{0,1\} \qquad \forall\, i,\ell \tag{5}$$

where d_i is the accumulation rate at customer site i and W is the vehicle capacity.

Let $\{(c^q, \phi^q, p^q, s^q)\}_{q\in Q}$ be the enumerated set of periodic routes, q, that are defined as the solutions, ϕ^q to the above knapsack subproblem along with their cost, c^q, and the definition of a periodicity p^q and a starting date, s^q. From the information given by (ϕ^q, p^q, s^q), one can generate an indicator matrix δ^q with $\delta^q_{it} = 1$ if the demand of period t for customer $i \in N$ is covered by route q and zero otherwise, while $\delta^q_{0\,t} = 1$ if the vehicle is used in period t and zero otherwise. Then, the inventory routing problem can be formulated as:

$$Z^d_{IP} = \min\ Vmax + \alpha \sum_{q\in Q} \frac{c^q}{p^q}\,\lambda_q \tag{6}$$

$$\sum_{q} \delta^q_{it}\lambda_q \geq 1 \qquad \forall i = 1,\ldots,n;\ t = 1,\ldots,T \tag{7}$$

$$\sum_{q} \delta^q_{0t}\lambda_q \leq Vmax \qquad \forall t = 1,\ldots,T \tag{8}$$

$$\lambda_q \in \{0,1\} \qquad \forall\, q \tag{9}$$

$$Vmax \in I\!N. \tag{10}$$

where $0 \leq \alpha < 1$ is a coefficient to balance both term in the objective, $\lambda_q = 1$ if periodic route q is used and zero otherwise, while $Vmax$ is the maximum number of vehicles used in a period. The variables λ_q and associated columns are generated dynamically in the course of the optimisation procedure (using a column generation approach).

The above formulation suffers from a symmetry in t: equivalent solutions can be defined that differ only by a permutation in the choice of starting dates. To avoid this drawback, we aggregate periods and model an average behavior. Technically speaking, we implement a state space relaxation in the space of the columns: aggregating all columns that differ only by their starting dates, we project our column space as follows

$$\{(c^q, \phi^q, p^q, s^q)\}_{q\in Q} \quad \underrightarrow{proj} \quad \{(c^r, \phi^r, p^r)\}_{r\in R}.$$

At each column r is associated a set of columns q such that r is the projection of q:

$$Q(r) = \{q : c^q = c^r, \phi^q = \phi^r, p^q = p^r, s^q \in \{1, \cdots, p^r\}\}\ .$$

While the former formulation is referred to as the *discrete time master* problem, the reformulation obtained after performing this mapping is called the *aggregate master*. It takes the form:

$$Z^a_{IP} = \min Vaver \quad + \quad \alpha \sum_{r \in R} \frac{c^r}{p^r} \lambda_r \tag{11}$$

$$\sum_{r \in R} \frac{\ell}{p^r} \phi^r_{i\ell} \lambda_r \geq 1 \qquad \forall i \tag{12}$$

$$\sum_{r \in R} \frac{1}{p^r} \lambda_r \leq Vaver \tag{13}$$

$$\lambda_r \in I\!N \qquad \forall\, r \tag{14}$$

$$Vaver \in I\!N. \tag{15}$$

where λ_r is the number of times that a vehicle uses periodic route r ($\lambda_r = \sum_{q \in Q(r)} \lambda_q$) and $Vaver$ is the average number of vehicles used per period.

We show that discrete and aggregate master program have the same optimal LP solution, but the solution of the aggregate master by column generation is much faster. Hence, we use the aggregate master to compute dual bounds. However, from an integer solution point of view, both formulations are not equivalent: the aggregate formulation is a relaxation of the problem. Hence, the discrete time formulation remains useful for computing primal bounds through heuristics.

3 Branch-and-Price-and-Cut

To obtain dual bounds, the aggregate master is solved to LP optimality by column generation. The pricing problem consist in generating a periodic route: the knapsack problem (1-5) is solved for each periodicity and each seed. A dynamic program [7] returns the knapsack solution. To speed up the resolution time, a preprocessing is performed at each step (items with negative reduced cost are not considered and the dynamic program is called only if there is hope to find a negative reduced cost column). Moreover, the enumeration of the pricing subproblems associated with each pair (periodicity, seed) stops as soon as a column with negative reduced cost is found.

A cutting planes procedure is implemented based on a family of valid inequalities that we derived from (12) using a rounding procedure. Let h be an integer ranging from 1 to $T-1$ and $i \in N$ such that $tmax_i > 1$, the inequalities take the form:

$$\sum_{r, h\ell \% p = 0} \frac{\ell}{p} \phi^r_{i\ell} \lambda_r + \sum_{r, h\ell \% p \neq 0} (\lceil \frac{h\ell}{p} \rceil - \frac{h\ell}{p}) \phi^r_{i\ell} \lambda_r \geq 1\ . \tag{16}$$

After each addition of a cut, we return to the column generation procedure.

Then, we further improve the dual bound through a truncated branch-and-bound procedure: we branch only on variable $Vaver$. Given the structure of our objective that focuses on vehicle use, this branching has an important impact of the bound. Moreover, the branch were $Vaver$ is rounded down can often be proved infeasible.

We have made comparative tests on real and randomly generated instances. We have 5 real instances with 60 customers on average, this group is named "S60", and 2 bigger instances with

172 and 157 customers, named GN172 and Pb157. We generate 10 random instances imitating the real problem (the customer coordinates are generated according to 3 schemes: urban, rural or mixed). This group is named "AL100" (we make these random instances available on our web site [11]). The dual bound improvement observed by adding cut is small (less than 2% in our numerical tests), but the improvement obtained through partial branching can get bigger (depending on the instance, it ranges from less than 1% up to more than 15%). In the Table 1, we present the dual bound obtained at the root of the branch-and-price tree, "rootDB", and the bound obtained after branching and adding cut, "DB+br+cut". To compare these bounds, we compute the gap using a good primal solution (see section 4), and give the gap at root, "gap", after adding cut, "gap+cut", after branching, "gap+br", and after branching and adding cut, "gap+br+cut". The initial dual bound is improved by 12.42%.

Name	rootDB	gap	gap+cut	gap+br	DB+br+cut	gap+br+cut
av AL100	325.674	22.81	21.31	10.37	366.123	9.24
av S60	224.836	26.38	25.51	12.51	254.083	11.83
GN172	517.348	16.46	15.59	8.65	558.452	7.89
Pb157	601	15.40	14.39	5.83	659.839	5.11
av 17 inst		**23.05**	**21.80**	**10.63**		**9.67**

Table 1: Dual bounds on our test bed

4 Heuristics based on column generation

To obtain primal bounds, we adapt several classical heuristics to the context of a column generation approach. In [6], we provide a classification of such methodologies and a review of previous work (f.i., [3, 4, 8, 9]) where greedy, local search, rounding or other LP based heuristics have been used in a decomposition approach.

A natural way to obtain an integer solution is to solve the master restricted to the set of generated columns as an integer program. For this, we must use the discrete master formulation (note that each column r generated for the aggregate master translates into a column q for each feasible starting date s in the discrete master). The rounding heuristic differs from the restricted master heuristic by the fact that new columns are generated in the course of the procedure but instead of fully exploring a branch-and-bound tree, a heuristic selection of a branch is made at each node: a column of the LP solution is rounded up and the master LP is re-optimized by column generation. For this implementation we use both the aggregate and the discrete master programs: LP solution are computed for the aggregate master; but the partial master solution is recorded in the discrete formulation (choosing a starting date for each column selected by the rounding procedure) and the columns used in the aggregate formulation are restricted to those that could be part of an integer solution to the residual discrete master program. Our third heuristic is a local search procedure where a neighbor solution is defined by removing a few columns from the current integer solution and re-building a complete integer solution with the rounding heuristic procedure.

Our computational experiments show that solving the restricted discrete master program to integer optimality is quite computationally intensive (for instances with 100 customers we have no solution after 2 hours of computing time); this is partially due to symmetry. With the rounding heuristic, we obtain primal bounds whose optimality gap is around 10% for instances of industrial

size. By imposing some restrictions on solution space (such as further restricting the set $P = \{1, 2, 3\}$) we sometime get smaller optimality gaps, a post-optimisation procedure gives the solution on the larger set $P = \{1, 2, 3, 6\}$. The local search procedure allows small improvements. In Table 2, we present the primal bound and the associated duality gap that we obtained by rounding heuristic called at root and after branching, "PB+RH" and "gap+RH", and by further applying the local search improvement procedure at the root node, "PB+LS" and "gap+LS", and the gap obtained in calling the rounding heuristics after restricting the set P to $\{1, 2, 3\}$ and making use of a post-optimisation heuristic, "gap+PO".

Name	PB+RH	gap+RH	PB+LS	gap+LS	gap+PO
av AL100	409.399	12.29	407.892	11.88	8.72
av S60	285.056	11.75	285.18	11.74	10.40
GN172	589.582	5.574	590.48	5.73	5.90
Pb157	731.48	10.85	708.594	7.39	5.11
av 17 inst		**11.66**		**11.22**	**8.84**

Table 2: Primal bounds on our test bed

5 Results for a large industrial instance

Finally, we applied our approach to a large real-life industrial instances with 260 customers. In 8h34m of computing time we obtained the dual bound, "DB+br+cut", of value 838.17 (5h21m was spent in the cutting plane procedure). If however one is only looking for a good primal solution, our primal heuristics can be run skipping the time consuming in cutting plane procedure. In a separate run (turning off the cutting plane procedure), we obtained in 2h53m a primal solution with gap of 6.23% (PB=890.384) by calling rounding heuristics followed by local search on restricted $P = \{1, 2, 3\}$ and by applying post-optimisation. If we compare this solution to the one that is currently used by our industrial partner, we observe that our solution requires 9 instead of 10 vehicles and reduces the travelling distance by more than 10 percent.

Conclusion

We consider a real-life application of the inventory routing problem on a tactical planning level. Our primal heuristics based on exact optimisation tools provides much better solutions than the greedy heuristic approaches that we originally tested. Our dual bounds allow us to provide a guarantee on the optimality gap. Moreover, comparison with the solution currently used by our industrial partner show significant improvements. The key to success is a formulation modelling an average behavior to avoid symmetry.

References

[1] J. Bramel, D. Simchi-Levi (1995), A location based heuristic for general routing problems, *Operations Research* **43**, 649 – 660.

[2] AM. Campbell, LW Clarke, MWP. Savelsbergh (2002), Inventory routing in practice, *The Vehicle Routing Problem*, eds P. Toth, D. Vigo, SIAM Monographs on Discrete Mathematics and Applications.

[3] A. Chabrier (2003), Heuristic Branch-and-Price-and-Cut to solve a network design problem, *Proceedings CPAIOR*.

[4] FL. Cimelire (2004), Optimisation du traitement de l'ordre de fabrication dans l'industrie textile, *Thesis, University Bordeaux 1*.

[5] V. Gaur, ML. Fisher (2004), A periodic inventory routing problem at a supermarket chain, *Operations Research* **52**, 813 – 822.

[6] S. **Michel** (2006). Optimisation des tournes de vhicules combine la gestion des stocks. Phd Thesis, University Bordeaux 1.

[7] D. Pisinger (1995), A minimal Algorithm for the Multiple-Choice Knapsack Problem, *EJOR* **83**, 394 – 410.

[8] E. Taillard (1999), A heuristic column generation method for the heterogeneous VRP, *RAIRO Operations Research* **33**, 1 – 14.

[9] F. Vanderbeck (2000), Exact Algorithm for minimising the number of setups in the one-dimensional cutting stock problem, *Operations Research* **48**, 915 – 926.

[10] F. Vanderbeck and M.W.P. Savelsbergh (2006), A Generic View of Dantzig-Wolfe Decomposition in Mixed Integer Programming, *Operations Research Letters* **34**(3), 296 – 306.

[11] IRP instances available at `http://www.math.u-bordeaux.fr/~smichel/IRPinstances.tar`

EDF Feasibility and Hardware Accelerators

Andrew Morton
University of Waterloo, Waterloo, Canada, arrmorton@uwaterloo.ca
Wayne M. Loucks
University of Waterloo, Waterloo, Canada, wmloucks@pads.uwaterloo.ca

A feasibility analysis is developed for embedded systems that use hardware accelerators to speed up critical portions of the software system. The scheduling policy analyzed is Earliest Deadline First. The concept of an accelerator-induced idle is introduced along with modifications in representation for tasks employing hardware accelerators. These modifications are incorporated into an existing algorithm for feasibility analysis of regular task sets. The changes introduced allow the benefits of the accelerators (parallel execution between processor and accelerator and reduced overall task execution time) to be accounted for, resulting in a less conservative analysis.

Keywords: Real-Time Scheduling, Theoretical Scheduling, Hardware Accelerators.

1 Introduction

This paper presents an algorithm for analyzing schedule feasibility under the Earliest Deadline First (EDF) policy. Specifically, EDF feasibility is examined for embedded real-time systems that use hardware accelerators to speed up parts of the software application.

For a hard real-time system, it is necessary that all tasks not only function properly but also meet their deadlines. Functional correctness without temporal correctness is failure and can have significant consequences. An algorithm to ensure schedule feasibility must be able to check that all tasks will meet their deadlines, every time.

The scheduling policy being examined in this paper is EDF. This is an important real-time scheduling policy for two reasons. First, it is optimal, in the sense that it will fail to meet a task's deadline only if no other scheduling policy could meet the deadline. Second, a task's priority is determined by the closeness of it's deadline. This is a more natural way to schedule real-time tasks than the rate monotonic policy, where fixed task priorities are set to approximate task deadlines.

The problem of analyzing schedule feasibility can be exacerbated in embedded systems by the use of hardware accelerators to speed up selected "kernels" of software that have high computation cost. In this paper, the feasibility analysis algorithm proposed by Stankovic *et al* [3] is extended for tasks employing hardware accelerators for speed-up.

In the next section, feasibility analysis for tasks sets scheduled by EDF is introduced. This introduction is followed by motivating the extended analysis for hardware accelerators and then deriving the analysis. Lastly, the implications and usability of the extended analysis are considered and conclusions are drawn.

2 EDF Schedule Analysis

Under the EDF policy, of all ready tasks, the task with the earliest deadline is executed first. If another task arrives with an earlier deadline, it will preempt the currently executing task. Periodic tasks occur at regular intervals. Periodic task τ_i has start time s_i, period T_i, relative deadline D_i and worst-case execution time C_i. τ_i is released at the start of each period at time $r_i = s_i + mT_i$,

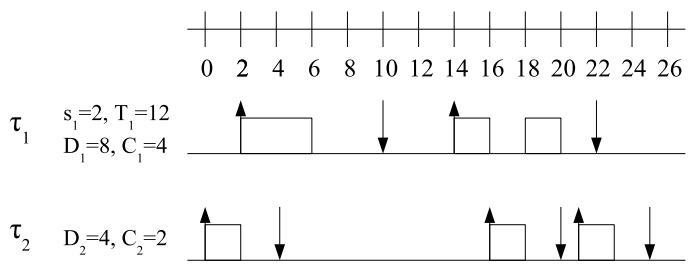


Figure 1: Task Notation

$m = 0, 1, \ldots$. The task must complete by absolute deadline $d_i = r_i + D_i$. Sporadic tasks are not released at regular intervals but can be characterized by minimum inter-arrival time T_i. For the purpose of analysis, the sporadic task can be conservatively represented by a periodic task. A task set composed of periodic and sporadic tasks is a *hybrid* task set.

In Figure 1, τ_1 is periodic and τ_2 is sporadic. Task releases are indicated with an up arrow and task deadlines are indicated with a down arrow. τ_1 has start time $s_1 = 2$, relative deadline $D_1 = 8$, period $T_1 = 12$ and worst-case execution time $C_1 = 4$. It has release dates $r_1 = 2, 14, \ldots$ and corresponding deadlines $d_1 = 10, 22, \ldots$. τ_2 is released sporadically at times $t = 0, 16, 21$. At $t = 16$ τ_2 preempts τ_1 because its deadline is closer. τ_1 is resumed at $t = 18$ after τ_2 terminates. The minimum inter-arrival time for τ_2, as shown in Figure 1, is $T_2 = 21 - 16 = 5$. An instance of a task is called a job. In Figure 1, τ_1 has 2 jobs and τ_2 has three jobs.

The purpose of EDF analysis is, given task set τ, determine if it can be proved that all n tasks in the set can be scheduled by the EDF policy such that no job misses its deadline. Stankovic *et al* [3] have developed an analysis for task sets, which is based on processor demand. It is summarized here.

2.1 Algorithm Summary

A necessary but not sufficient condition for the feasibility of a task set scheduled by the EDF policy is that the processor utilization U (defined by Liu and Layland [1]) does not exceed one:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1}$$

U sums the fraction of processor time required per task.

The rest of the analysis assumes that the task set is synchronous: all tasks are released together at $t = 0$ ($\forall i$: $s_i = 0$). This is the most constraining scenario, otherwise known as the "critical instance". After checking $U \leq 1$, the algorithm checks processor demand. Processor demand, $h(t)$, in the interval $[0, t)$ is defined:

$$h(t) = \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i. \tag{2}$$

Processor demand at time t measures the work required by all tasks having deadlines in $[0, t]$. Processor demand is applied in the following theorem:

Theorem 1 (Processor Demand Condition [3])**.** *Any given hybrid task set is feasible under EDF scheduling* if *and* only if

$$\forall t: \quad h(t) \leq t.$$

However, it is not necessary to test processor demand on all intervals in $[0, t)$; it is sufficient to test only when task deadlines occur [4]. Furthermore, an upper bound can be placed on the interval over which $h(t)$ is checked. This is done by calculating the length L of the synchronous busy period[1].

Theorem 2 (Liu and Layland [1])**.** *When the deadline driven scheduling algorithm is used to schedule a set of tasks on a processor, there is no processor idle time prior to an overflow.*

The interval of time preceding the first idle period is called the synchronous busy period. Stankovic *et at* describe the following iterative method for calculating the length L of the synchronous busy period. This algorithm converges in $O(\sum_{i=1}^{n} C_i)$ time, if $U < 1$ [2].

Apply recursively until $L^{m+1} = L^m$:

$$\begin{cases} L^{(0)} &= \sum_{i=1}^{n} C_i, \\ L^{(m+1)} &= W(L^{(m)}), \end{cases} \tag{3}$$

where

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i. \tag{4}$$

In essence, all tasks are released at $t = 0$ and their cumulative execution times (busy period) are calculated $L^{(0)}$. The algorithm iteratively checks how many jobs have been released in $L^{(m)}$ and sums their execution times $L^{(m+1)}$. When it reaches $L^{(m+1)}$ where all tasks have finished executing before any new jobs are released, it terminates.

The set S of deadline events is defined over $[0, L)$, and for each event v in S, it is checked that the processor demand doesn't exceed processor time ($h(v) \leq v$). Algorithm 1 lists the pseudocode for Stankovic *et al*'s feasibility analysis algorithm.

Algorithm 1: EDF Feasibility Analysis

```
if U > 1 then return "not feasible" ;
S = ∪_{i=1}^{n} {mT_i + D_i : m = 0, 1, ...} = {v_1, v_2, ...} // deadlines ;
k ← 1 ;
while v_k < L do
    if h(v_k) > v_k then return "not feasible" // check processor demand ;
    k ← k + 1 ;
end
return "feasible" ;
```

3 Hardware Accelerators and Scheduling

A common platform for embedded systems is the System on Chip (Soc). In a SoC, a general purpose processor (CPU), RAM, ROM, I/O and application specific hardware are integrated on one IC. The application specific hardware typically consists of accelerators for portions, otherwise known as "kernels", of the application that do not execute fast enough in software or consume too much processor time. Accelerators that execute independently of any software task are not considered here as they do not pose a problem for schedule analysis. However a software task that invokes a hardware accelerator and then waits, or "blocks" until the accelerator finishes before continuing is

(a) Software Task τ_i (b) Hardware Accelerated τ_i

Figure 2: Accelerator-Blocked Task

Table 1: Accelerator-Blocked Task Division

Task	Target	Period	Deadline	Time
$\tau_{i,1}$	proc.	$T_{i,1} = T_i$	$D_{i,1} = D_{i,2} - \vec{C}_{i,2}$	$C_{i,1}$
$\tau_{i,2}$	hw	$T_{i,2} = T_i$	$D_{i,2} = D_{i,3} - \vec{C}_{i,3}$	$C_{i,2}$
$\tau_{i,3}$	proc.	$T_{i,3} = T_i$	$D_{i,3} = D_i$	$C_{i,3}$

of interest for schedule analysis. Such a software task is called an "accelerator-blocked" task and is illustrated in Figure 2.

The task τ_i has three parts, $\tau_{i,1}, \tau_{i,2}, \tau_{i,3}$. Part $\tau_{i,2}$ has been identified for speed-up in hardware. In Figure 2(a) it executes in software and in Figure 2(b) $\tau_{i,2}$ is replaced by a hardware accelerator. A simple approach to the analysis is to calculate the worst-case execution time of task τ_i as the sum $C_i = C_{i,1} + C_{i,2} + C_{i,3}$ and use Algorithm 1 without modification. This may be a reasonable approach when $C_{i,2} << C_{i,1} + C_{i,3}$. The primary drawback of this approach is that the parallel execution between processor and accelerator is ignored. In other words, while τ_i is waiting for $\tau_{i,2}$ to execute on the hardware accelerator, another task could be using the processor. The algorithm could rule a task set infeasible that is actually feasible. The goal of the extended analysis is to account for the parallel execution between the processor and hardware accelerators.

4 Extended Analysis

The analysis is initially developed for a task set in which one task blocks on an accelerator once during its execution. Extension to a task set in which one task blocks multiple times on an accelerator during its execution will then be briefly discussed.

To aid in the extended analysis, the execution time of each task is represented by vector $\vec{C}_i$. For a regular task the size of the vector is $|\vec{C}_i| = 1$. For the accelerator-blocked task from Figure 2(b), $|\vec{C}_i| = 3$. $\vec{C}_{i,2}$ is execution time on the accelerator.

In order to perform the analysis, the accelerator-blocked task τ_i is replaced by three ordered subtasks as described in Table 1. The deadlines are modified so that $\tau_{i,1}$ finishes in time for $\tau_{i,2}$ and $\tau_{i,3}$ to finish. Likewise, $\tau_{i,2}$ finishes in time for $\tau_{i,3}$ to finish. Because EDF schedules tasks with earlier deadlines first, the subtasks will execute in the correct order.

The task set τ is modified by replacing the accelerator-blocked task τ_i with its two software subtasks $\tau_{i,1}$ and $\tau_{i,3}$. For now $\tau_{i,2}$, which executes in hardware, is ignored. It will be accounted for later.

Now consider the example task set in Figure 3. If $\tau_{1,1}$, $\tau_{1,3}$ and τ_2 are used as the task set in the analysis of Algorithm 1, it will pass the analysis and be deemed feasible. That is because there is enough processor time to meet the processor demand at all deadline events $S = \{2, 7, 11\}$ in the synchronous busy period. The problem arises with the second jobs of τ_1 and τ_2. There is enough processor time to fulfill the software tasks' demands but $\tau_{i,3}$ cannot use some of the available time

[1] In [3], the upper bound is chosen from the minimum of three values. The other two upper bounds are omitted here as they are not used in the extended algorithm in Section 4.

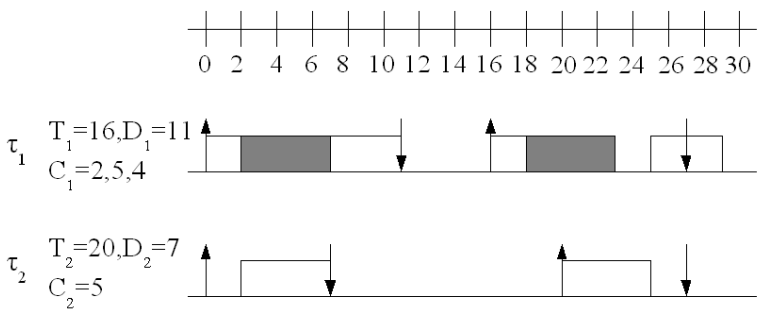


Figure 3: Feasibility Failure

because it is waiting for $\tau_{i,2}$ to finish on the accelerator. This causes an "accelerator-induced" idle that is not accounted for by the analysis. The goal of the extended analysis is to determine whether an accelerator-induced idle can result in a missed deadline.

Lemma 1. *Given a task set τ in which one task τ_x blocks on an accelerator once, with logical subtasks $\{\tau_{x,1}, \tau_{x,2}, \tau_{x,3}\}$, the accelerator-induced idle results in a critical instance when:*

1. *$\tau_{x,3}$ has no slack, and*
2. *all other tasks are released synchronously at the end of the accelerator-induced idle (i.e. at $t = D_{x,2}$).*

A second critical instance has been introduced. The first is the synchronous release of all tasks in τ. The second is an accelerator-induced idle during which no other tasks are ready to execute but at the end of which all tasks (except $\tau_{x,1}$) are released.

Before proving Lemma 1, loading factor must be introduced. Loading factor on the interval $[t_1, t_2)$ is defined as

$$u_{[t_1,t_2)} = \frac{h_{[t_1,t_2)}}{(t_2 - t_1)}.$$

$h_{[t_1,t_2)}$ is the sum of work released not earlier than t_1 with deadline not later than t_2:

$$h_{[t_1,t_2)} = \sum_{t_1 \le r_k, d_k \le t_2} C_k,$$

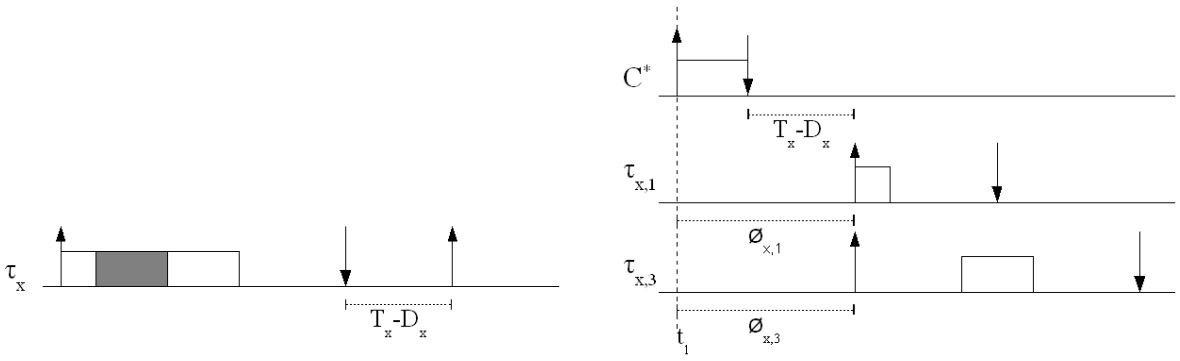
where r_k, d_k and C_k are release, deadline and worst-case execution time for jobs in that interval. Loading factor is the fraction of the interval required to meet its processor demand. The absolute loading factor is the maximum loading factor of all possible intervals:

$$u = \sup_{0 \le t_1 < t_2} u_{[t_1,t_2)}.$$

The proof of Lemma 1 now follows.

Proof. The worst-case schedule feasibility occurs when:

- **Part 1:** $\tau_{x,3}$ has no slack ($d_{x,3} = \vec{C}_{x,3}$)
  If $\tau_{x,3}$ has no slack, then no other work can be done before $d_{x,3}$. (i.e. if there exists another task with deadline before $d_{x,3}$, then at $t = d_{x,3}$, $h(t) > t$ which makes the schedule infeasible.) Adding slack would only improve schedule feasibility.



(a) τ_x (b) $C^*, \tau_{x,1}, \tau_{x,3}$

Figure 4: Decomposition of τ_x

- **Part 2:** all other tasks are released synchronously with $\tau_{x,3}$
  The remaining part of this proof is the same, in essence, as the proof of Lemma 3.1 in [3] except for modifications to the accelerator-blocked task as described next.

  Let τ be the asynchronous task set $(\exists i \mid s_i \neq 0)$ and τ' the corresponding synchronous task set and let the accelerator-blocked task, τ_x, be excluded from τ and τ'. When the processor demands, $h_{[t_1,t_2)}$ and $h'_{[t_1,t_2)}$, of τ and τ' are computed, they are augmented by the accelerator-blocked task as follows:

  – In calculating the processor demand, a unit of work C^* is released at t_1 with duration $C^* = \vec{C}_{x,3}$, representing the part of the accelerator-blocked task that must execute after the accelerator-induced idle. The deadline of C^* is D^* which is equal to $\vec{C}_{x,3}$ (no slack). Furthermore τ_x is replaced by $\tau_{x,1}$ and $\tau_{x,3}$. Since C^* represents the remaining work of the previous job of τ_x which has a deadline at $t = \vec{C}_{x,3}$, the next job of τ_x is released at $T_x - D_x + \vec{C}_{x,3}$ after t_1. Tasks $\tau_{x,1}$ and $\tau_{x,3}$ are said to have phases $\phi_{x,1} = \phi_{x,3} = T_x - D_x + \vec{C}_{x,3}$. Only an accelerator-blocked task is said to have phase; for all other tasks $\phi_i = 0$ (note that phase and start time are not related). Figure 4 shows the relationship between τ_x and C^* and $\tau_{x,1}$ and $\tau_{x,3}$ with phases.

  In order to prove that synchronous release of all other tasks but the accelerator-blocked task is the worst-case scenario, it is sufficient to prove that the absolute loading factor ("loading factor") u of τ is bounded above by the loading factor u' of τ'. It is therefore sufficient to prove that $\forall[t_1, t_2)$ there exists an interval $[t'_1, t'_2)$ such that $h_{[t_1,t_2)} \leq h'_{[t'_1,t'_2)}$, where the accelerator-blocked task is included as described in the previous paragraph. For any interval $[t_1, t_2)$, the processor demand for τ is bounded above by:

$$h_{[t_1,t_2)} \leq C^* + \sum_{\substack{D_i+\phi_i \\ \leq t_2-t_1}} \left(1 + \lfloor \frac{t_2 - t_1 - (D_i + \phi_i)}{T_i} \rfloor\right) C_i.$$

  Addition of ϕ_i to this equation only affects $\tau_{x,1}$ and $\tau_{x,3}$.

  Now consider the synchronous case, τ', and let $t'_1 = 0$ and $t'_2 = t_2 - t_1$. Processor demand is

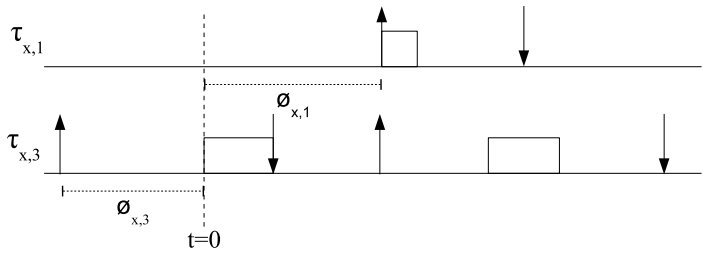


Figure 5: $\phi_{x,1}$ and $\phi_{x,3}$

equal to:

$$h'_{[t'_1,t'_2)} = C^* + \sum_{\substack{D_i+\phi_i \\ \leq t'_2-t'_1}} \left(1 + \lfloor \frac{t'_2 - t'_1 - (D_i+\phi_i)}{T_i} \rfloor\right) C_i$$
$$= C^* + \sum_{\substack{D_i+\phi_i \\ \leq t_2-t_1}} \left(1 + \lfloor \frac{t_2 - t_1 - (D_i+\phi_i)}{T_i} \rfloor\right) C_i.$$

Processor demand of the synchronous case, $h'_{[t'_1,t'_2)}$, equals the upper bound of the non-synchronous case, $h_{[t_1,t_2)}$. Therefore:

$$h_{[t_1,t_2)} \leq h'_{[t'_1,t'_2)}.$$

□

To apply this result, the analysis of Algorithm 1 is done twice: once for each critical instance (synchronous release and accelerator-induced idle). The synchronous busy period and processor demand calculations need to be redefined for the second critical instance. To facilitate this, the phase of the accelerator-blocked task is redefined.

In the proof above, the portion of the accelerator-blocked task remaining after the accelerator-induced idle was represented by C^* and the next job of the task had phase $\phi_{x,1} = \phi_{x,3} = T_x - D_x + C_{x,3}$. This can also be represented by modifying the phase of $\tau_{x,3}$: $\phi_{x,3} = -(D_x - C_{x,3})$. By doing this $\tau_{x,3}$ has an initial deadline at $t = C_{x,3}$. Since $D_x - C_{x,3} = D_{x,2}$, the phases of the two subtasks of the accelerator-blocked task τ_x become:

$$\phi_{x,1} = T_x - D_{x,2} \qquad \phi_{x,3} = -D_{x,2},$$

and every other task has phase $\phi_i = 0$. The change in $\phi_{x,3}$ can be seen by comparing Figures 4 and 5. In Figure 5, C^* has been incorporated into $\tau_{x,3}$.

The synchronous busy period is now calculated as follows:

Apply recursively until $L'^{m+1} = L'^{m}$:

$$\begin{cases} L'^{(0)} &= \sum_{\phi_i \leq 0} C_i, \\ L'^{(m+1)} &= W'(L'^{(m)}), \end{cases} \tag{5}$$

where

$$W'(t) = \sum_{\phi_i \leq t} \left\lceil \frac{t - \phi_i}{T_i} \right\rceil C_i. \tag{6}$$

Figure 6: One Task, Multiple Blocks

When $\phi_{x,1} = \phi_{x,3} = 0$, then $L' = L$. When $\phi_{x,1} = T - D_{x,2}$ and $\phi_{x,3} = -D_{x,2}$, then the result is different: at $t = 0$ all of the regular tasks are released as well as accelerator-blocked subtask $\tau_{x,3}$. The processor demand calculation is also modified for phase:

$$h'(t) = \sum_{D_i+\phi_i \leq t} \left(1 + \left\lfloor \frac{t - (D_i + \phi_i)}{T_i} \right\rfloor\right) C_i. \tag{7}$$

4.1 Algorithm Analysis

Algorithm 1 is run twice: once with $\phi_{x,1} = \phi_{x,3} = 0$, and once with $\phi_{x,1} = T_x - D_{x,2}$ and $\phi_{x,3} = -D_{x,2}$. In both instances, τ_x is replaced by $\tau_{x,1}$ and $\tau_{x,3}$ in τ. As a result, the execution on the hardware accelerator is omitted from the processor demand analysis. The implication is that the parallel execution between processor and accelerator is incorporated into the feasibility analysis. However to do this, a second critical instance is added that looks at the worst-case scenario that can follow an accelerator-induced idle. This puts a limitation on the type of task set that will pass the analysis: all regular tasks must have enough slack to permit $\tau_{x,3}$ to execute at $t = 0$. Since accelerator execution times are generally shorter than task execution times, it is expected that this limitation will not be onerous.

Extending the algorithm to task sets in which one task blocks multiple times is described only briefly due to space limitations. The length of the execution time vector $\vec{C}_x$ of the accelerator-blocked task would be incremented by two for each {accelerator,software} pair added to it's execution as in Figure 6. Each time the task blocks on a accelerator, it may result in a accelerator-induced idle. Thus each accelerator block in the vector is associated with a critical instance. The analysis of Algorithm 1 is performed once with all phases equal zero and then is repeated once per block with phases modified to simulate the appropriate accelerator-induced idle.

5 Conclusions

Task sets in which a task blocks on an accelerator have been shown to add a second critical instance for feasibility analysis. This is done by including only the parts of that accelerator-blocked task that execute on the processor in τ. Phases for this subtasks were introduced to facilitate analysis of the second critical instance. The analysis incorporates the parallel execution between processor and accelerator but impose some limitations on the characteristics of task sets that can pass the analysis. Analysis of task sets with multiple accelerator-blocked tasks is left for future work.

References

[1] C. L. Liu and J. W. Layland (1973), Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM* **20**(1), 46 – 61.

[2] I. Ripoll, A. Crespo, and A. K. Mok (1996), Improvement in feasibility testing for real-time tasks, *Real-Time Systems* **11**(1), 19 – 39.

[3] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo (1998), *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*, Kluwer Academic Publishers.

[4] Q. Zheng and K. G. Shin (1994) On the ability of establishing real-time channels in point-to-point packet-switched networks, *IEEE Transactions on Communications* **42**(2).

An Approximation Algorithm for the UET Two-Machine Open-Shop Problem with Time Delays

Alix Munier-Kordon
Laboratoire LIP6, Université Pierre et Marie Curie, 4 place Jussieu, 75 252, Paris Cedex 05, France,
Alix.Munier@lip6.fr

Djamal Rebaine
Département d'informatique et de mathématique, Université du Québec à Chicoutimi, 555, Boul. de l'Université, Québec, Canada G7H 2B1, drebaine@uqac.ca

This paper addresses the problem of scheduling n unit execution time (UET) jobs with time delays considerations on a two-machine open-shop environment. The criterion we are considering is the makespan. A simple approximation algorithm, with an asymptotic performance guarantee of $\frac{5}{4}$, is presented and proved.

Keywords: Makespan, Open-shop, Theoretical Scheduling, Time delays.

1 Introduction

This paper addresses the problem of scheduling n unit execution times (UET) jobs with time delays considerations on a two-machine open-shop environment. Each job comprises two operations. One operation has to be processed by one machine and the other by the second machine. As usual, we assume that the operations of a job cannot be processed at the same time, and a machine can only process at most one operation at a time.

In this study, we suppose that a time delay, τ_i, is associated with each job $i \in J$ to denote the minimum time which must elapse between the completion of one of its operation and the start of the other operation. In other words, if we denote by S_i and C_i, respectively, the start time of the second operation and the finish time of the first operation of job i, then a schedule to be valid must be such that $S_i \geq C_i + \tau_i,$ for $i = 1, \ldots, n$. The purpose is to find a valid schedule which minimises the overall completion time criterion, known as the makespan.

Motivation for the formulated problem comes from real applications. In the traditional shop scheduling, it is a common practice to assume that once a job has finished one of its operation, it becomes immediately available for further processing. However, in many applications, this assumption is not justified. Indeed, there is often a significant time delay between the completion of an operation and the beginning of the next operation of the same job. In addition to delineating the borderline between polynomiality and intractability, in some cases, the execution times might even be negligible compared to the time delays: assuming in this case that the processing times of the jobs are unitary, and therefore have a small influence on the makespan of the schedule. Time delays may be attributed, for example, to transportation times of the jobs on the machines or, in some other applications, to the different times needed by the drying processes of the jobs before they can be handled by another processing stage.

Open-shop problems with time delays may be used to model for example the timetable problems in which students have to meet each of their professors. A time delay refers in this case to the time taken by a student to go from one professor to another. More real world applications may be found in Gonzalez [1] and Lopez and Roubellat [3].

The open-shop scheduling problem with time delays was first introduced in Rayward-Smith and Rebaine [4], and shown that the corresponding two-machine problem is $\mathcal{NP}$-hard even for identical time delays. A further result is given in Yu [2] as the unary $\mathcal{NP}$-hardness is shown for the UET case.Therefore, looking for well solvable cases and heuristic algorithms is well justified.

The search for well solvable cases implies the search for special values of time delays for which the corresponding problem can be solved to optimality in polynomial time. Whereas, in the heuristic approach, we usually seek algorithms for which we can evaluate their worst case performance by measuring the distance between the value of the solution it generates and the optimal value. In the case of two machines and arbitrary processing times, Strusevich [6] developed an $O(n \log n)$ 3/2-approximation algorithm. In Rebaine and Strusevich [5], an $O(n)$ 4/3-approximation algorithm is presented for the case of two values of time delays: a time delay for the set of jobs going from the first to the second machine, and another time delay for the other direction. When the time delays are smaller than the processing times, then it is shown in the latter paper that the corresponding problem is solvable in linear time.

This paper is devoted to the development of an approximation algorithm for the UET two-machine open-shop with arbitrary integral time delays. It is organized as follows. In Section 2, we present preliminary results. Section 3 presents the Sorting Algorithm, and a special case for which it is optimal. The proof of the 5/4-asymptotic performance guarantee of this algorithm is presented in Section 4. Section 5 is our conclusion.

2 Preliminary results

In this section, we present results needed for the next sections. First, we introduce the following definition.

Definition 1. *For each job, $j \in J$, a valid schedule is going to process one operation earlier than the other - such an operation is called a first operation. The remaining operation is a second operation.*

Lemma 1. *There exists an optimal schedule in which the completion time of any first operation on each machine is not greater than that of any second operation*

Proof. Assume that a machine processes the second operation of job k immediately before the first operation of job j. These can be interchanged still obtaining a valid schedule. Repeating this argument establishes the lemma. □

Lemma 2. *There exists an optimal schedule in which the first operations and the second operations on each machine are processed continuously.*

Proof. Let us consider an optimal schedule satisfying Lemma 1. Assume that there exists an idle time between two consecutive first operations i and j, in that order. It is clear that j can be shifted to the left so as there is no idle time between i and j without increasing the makespan. Similarly, assume that there exists an idle time between two consecutive second operations i and j, in that order. It is clear that i can be shifted to the right so as there is no idle time between i and j without increasing the makespan. Repeating this argument establishes the lemma. □

Lemma 3. *The makespan of an optimal schedule, ω_{opt}, satisfies the following lower bound*

$$\omega_{opt} \geq \left\lceil \frac{\sum_{i=1}^{n} \tau_i}{n} \right\rceil + \left\lceil \frac{n}{2} \right\rceil .$$

Proof. Without loss of generality, we may assume that an optimal schedule satisfies Lemmas 1 and 2. Let T_1, with cardinality n_1 (respectively T_2, with cardinality n_2), be the set of jobs processed first on machine 1 (respectively 2) and then on machine 2 (respectively 1) in an optimal schedule. Let us first observe that $n_1 + n_2 = n$. Let also σ_1 (respectively σ_2) and π_1 (respectively π_2) be the job processing sequences of T_1 (respectively T_2) on machine 1 (respectively 2) and machine 2 (respectively 1), respectively, as illustrated by Figure 1.

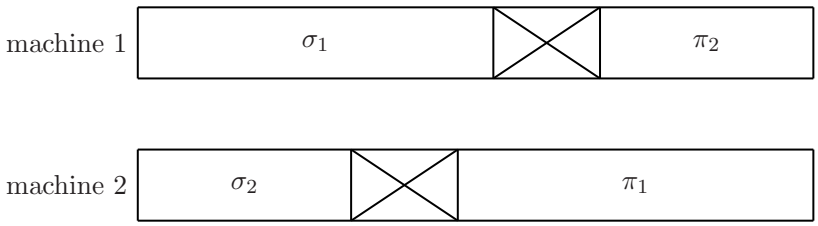


Figure 1: The permutations σ_1, σ_2, π_1 and π_2

To be valid, this schedule must be such that

$$\omega_{opt} - n_1 + \pi_1(j) - \sigma_1(j) \geq \tau_j + 1; \text{ for all } j \in T_1. \tag{1}$$

If we consider T_2 then by symmetry we get the following inequality:

$$\omega_{opt} - n_2 + \pi_2(j) - \sigma_2(j) \geq \tau_j + 1; \text{ for all } j \in T_2. \tag{2}$$

Now, if we add up the n_1 inequalities of (1), we get the new following inequality:

$$n_1\omega_{opt} - n_1^2 + \sum_{j\in T_1} \pi_1(j) - \sum_{j\in T_1} \sigma_1(j) \geq \sum_{j\in T_1} \tau_j + n_1. \tag{3}$$

Similarly, if we add up the n_2 inequalities of (2), we end up with the new following inequality:

$$n_2\omega_{opt} - n_2^2 + \sum_{j\in T_2} \pi_2(j) - \sum_{j\in T_2} \sigma_2(j) \geq \sum_{j\in T_2} \tau_j + n_2. \tag{4}$$

As $\pi_1(j)$ and $\sigma_1(j)$ for T_1 and $\pi_2(j)$ and $\sigma_2(j)$ for T_2 are permutations whose values are in $\{1, \ldots, n_1\}$ and $\{1, \ldots, n_2\}$, respectively, it follows that

$$\begin{aligned} \sum_{j\in T_1} \pi_1(j) - \sum_{j\in T_1} \sigma_1(j) &= 0, \\ \sum_{j\in T_2} \pi_2(j) - \sum_{j\in T_2} \sigma_2(j) &= 0. \end{aligned}$$

Now, if we add up inequalities of (3) with inequalities of (4), then we obtain the following:

$$n\omega_{opt} \geq \sum_{i=1}^{n} \tau_i + n + n_1^2 + n_2^2.$$

As $n_1 = n - n_2$, it is easy to see that $n_1^2 + n_2^2$ has its minimum at $n_1 = \dfrac{n}{2}$. Therefore,

$$\omega_{opt} \geq \frac{\sum_{i=1}^{n} \tau_i}{n} + 1 + \frac{n}{2}.$$

It then follows that

$$\begin{aligned} \omega_{opt} &\geq \left\lceil \frac{\sum_{i=1}^{n} \tau_i}{n} + 1 + \frac{n}{2} \right\rceil \\ &\geq \left\lceil \frac{\sum_{i=1}^{n} \tau_i}{n} \right\rceil + \left\lceil \frac{n}{2} \right\rceil. \end{aligned}$$

Thus the result. □

Lemma 4. *The makespan of an optimal schedule ω_{opt}, satisfies the following lower bound.*

$$\omega_{opt} \geq n.$$

Proof. As each of the n UET jobs is processed by each machine, the lemma follows immediately. □

3 Presentation of the Sorting Algorithm

In this section, we present a polynomial time algorithm, the Sorting Algorithm, that solves approximately the UET two-machine open-shop with time delays. We also present a simple case for which the Sorting Algorithm generates an optimal solution.

The Sorting Algorithm can be described as follows. First, the jobs are sorted in non-increasing order of the time delays. Then, the first operations are processed using a list scheduling algorithm starting with machine 1. Next, the second operations are executed using again a list scheduling algorithm. The running time of this algorithm is clearly dominated by the sorting procedure which can be implemented in $O(n \log n)$.

Sorting Algorithm

1. rename the jobs such that $\tau_1 \geq \tau_2 \geq \cdots \geq \tau_n$;
2. for $(i = 1; i <= n; i++)$
   schedule job i on the first available machine starting with machine 1;
3. for $(i = 1; i <= n; i++)$
   schedule job i as soon as possible on the corresponding machine.

For example, let us consider an instance of the problem defined by $n = 8$ jobs with $\tau_1 = \tau_2 = 7$, $\tau_3 = \tau_4 = 6$, $\tau_5 = 4$, $\tau_6 = 3$ and $\tau_7 = \tau_8 = 1$. The schedule obtained by the Sorting Algorithm is pictured by Figure 2.

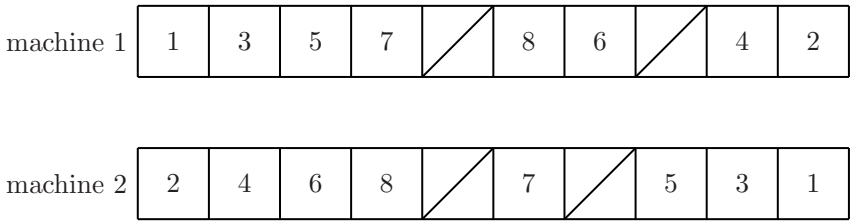


Figure 2: A schedule built by the Sorting Algorithm

Theorem 1. *If the time delays are distinct, then the Sorting Algorithm generates an optimal solution.*

Proof. Let us first observe that the shape of the schedule generated by the Sorting Algorithm satisfies the shape of the schedule of Lemma 1. Now, consider the first operation of two consecutive jobs i and j, in this order, on one of the two machines. We have that $\tau_i \geq \tau_j$. Moreover, as the time delays are all distinct, then clearly we have that $\tau_i \geq \tau_j - 2$. It follows that the start time of the second operation of job i is strictly greater than the start time of the second operation of job j. Therefore, as job 1 is processed first on machine 1, then the makespan of the schedule generated by the Sorting algorithm is clearly $C_{\max} = \tau_1 + 2$, which is a simple lower bound on the makespan. Thus the result. □

4 Performance ratio of the Sorting Algorithm

In this section, we present the worst-case performance of the Sorting Algorithm. To do so, we first discuss two special cases, then we proceed with the general case.

4.1 First special case

Let r be a positive integer such that $r \geq \lceil \frac{n}{2} \rceil$. We suppose here that I_1 is an instance of the UET two-machine open-shop problem such that, using the Sorting Algorithm, the second operations are available at time r. More formally, for any $i \in \{1, \ldots, n\}$, we have that $\tau_i = r - \lceil \frac{i}{2} \rceil$. Observe that the makespan of the schedule obtained by the Sorting Algorithm is $\omega_A(I_1) = r + \lceil \frac{n}{2} \rceil$. Let us first proceed with the following technical result.

Lemma 5.

$$\sum_{j=1}^{n} \left\lceil \frac{j}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil \left(n - \left\lceil \frac{n}{2} \right\rceil + 1 \right).$$

Proof. First, observe that

$$\sum_{j=1}^{n} \left\lceil \frac{j}{2} \right\rceil = 2 \sum_{\alpha=1}^{\lceil \frac{n}{2} \rceil} \alpha + \left\lceil \frac{n}{2} \right\rceil \left(n - 2 \left\lceil \frac{n}{2} \right\rceil \right).$$

Therefore,

$$\begin{aligned} \sum_{j=1}^{n} \left\lceil \frac{j}{2} \right\rceil &= \left\lceil \frac{n}{2} \right\rceil \left(\left\lceil \frac{n}{2} \right\rceil + 1 \right) + \left\lceil \frac{n}{2} \right\rceil \left(n - 2 \left\lceil \frac{n}{2} \right\rceil \right) \\ &= \left\lceil \frac{n}{2} \right\rceil \left(n - \left\lceil \frac{n}{2} \right\rceil + 1 \right). \end{aligned}$$

Thus the result. □

Lemma 6.

$$\frac{\sum_{j=1}^{n} \tau_j}{n} \geq r - \frac{3}{2} - \frac{n}{4}.$$

Proof. By definition, we have that

$$\sum_{i=1}^{n} \tau_i = nr - \sum_{i=1}^{n} \left\lceil \frac{i}{2} \right\rceil.$$

Now, if we use Lemma 5, then we get the following:

$$\begin{aligned}\sum_{i=1}^{n}\tau_i &= nr - \left\lceil\frac{n}{2}\right\rceil\left(n - \left\lceil\frac{n}{2}\right\rceil + 1\right)\\ &\geq nr - n\left(\frac{n+1}{2}\right) + \left(\frac{n}{2}\right)^2 - \left(\frac{n+1}{2}\right).\end{aligned}$$

Therefore,

$$\begin{aligned}\sum_{i=1}^{n}\tau_i &\geq nr - \frac{(n+1)^2}{2} + \left(\frac{n}{2}\right)^2\\ &= nr - \frac{n^2}{4} - n - \frac{1}{2}.\end{aligned}$$

Since $n \geq 1$, then the result follows. □

Theorem 2.

$$\frac{\omega_A(I_1)}{\omega_{opt}(I_1)} \leq \frac{5}{4} + \frac{3}{2n}.$$

Proof. From Lemmas 3 and 6, we deduce that

$$\begin{aligned}\omega_{opt}(I_1) &\geq r - \frac{3}{2} - \frac{n}{4} + \left\lceil\frac{n}{2}\right\rceil\\ &= \omega_A(I_1) - \frac{3}{2} - \frac{n}{4}.\end{aligned}$$

From Lemma 4, it follows that

$$\omega_A(I_1) \leq \frac{5}{4}\omega_{opt}(I_1) + \frac{3}{2}.$$

Thus the result. □

We now show that the above bound is tight. Let k be a strictly positive integer. Consider an instance of the problem defined by $n = 4k + 2$ jobs with, for any job $i \in \{1, \ldots, n\}$, $\tau_i = 3k - \lceil\frac{i}{2}\rceil$. The length of the schedule obtained by the Sorting Algorithm is clearly $\omega_A = 5k + 1$. Now, an optimal schedule with no idle time slots may be built by processing the first operation of every odd (respectively even) job by machine 1 (respectively 2). Jobs with odd delays are processed first on both machines in decreasing order followed by those with even delays in decreasing order. An optimal schedule for $k = 3$ is pictured by Figure 3. This gives a schedule of length $\omega_{opt} = 4k + 2$. Thus the tightness of the above bound.

machine 1	3	7	11	1	5	9	13	12	8	4	14	10	6	2
machine 2	4	8	12	2	6	10	14	11	7	3	13	9	5	1

Figure 3: An optimal schedule for the first special case with $k = 3$

4.2 Second special case

We now consider a second instance I_2 of the above problem such that, using the Sorting Algorithm, the second operation of jobs processed first by machine 1 (respectively 2) are available at time r (respectively $r-1$). More formally, for any $i \in \{1,\ldots,n\}$, we set $\tau_i = r - \lceil \frac{i+1}{2} \rceil$. We observe that $\omega_A(I_2) = r + \lceil \frac{n}{2} \rceil$. Proofs of Lemma 7 and Theorem 3 are similar to the previous case and are thus omitted.

Lemma 7.

$$\frac{\sum_{j=1}^{n} \tau_i}{n} \geq r - \frac{11}{4} - \frac{n}{4}.$$

Theorem 3.

$$\frac{\omega_A(I_2)}{\omega_{opt}(I_2)} \leq \frac{5}{4} + \frac{11}{4n}.$$

4.3 General case

Let I be an instance of the UET two-machine open-shop with $\omega_A(I) > n$. Let V_1 (respectively V_2) be jobs from I for which the first (respectively second) operations are processed by machine 1 (respectively 2) according to the Sorting Algorithm. Let us define the release date, r_i, of the second operation of job $i \in \{1,\ldots,n\}$ by $r_i = \tau_i + \lceil \frac{i}{2} \rceil$.

Let $r^\star$ be the smallest integer such that there is no idle time slots in the interval $[r^\star, \omega_A(I)]$. If there is an idle time slot at time $r^\star - 1$ on machine 1, we set $k^\star = 2$, otherwise $k^\star = 1$.

Lemma 8. *If $M = \{j \in V_{k^\star}, r_j \geq r^\star\}$ then $\omega_A(I) = r^\star + |M|$.*

Proof. The second operation of jobs from M are available after time $r^\star$. Moreover, since there exists an idle time slot at time $r^\star - 1$, every job $j \in V_{k^\star}$ with $r_j < r^\star$ is completed at time $r^\star - 1$. Thus the result. □

Let $t_1,\ldots,t_k$, $k = |M|$, be the successive completion times of the first operations of jobs from M. For an illustrative purpose, let us go back to the example pictured by Figure 2. We have that $r^\star = 8$, $M = \{2,4\}$, and $k^\star = 2$. The completion times are $t_1 = 1$ and $t_2 = 2$.

Lemma 9. *For any $\alpha \in \{1,\ldots,k\}$, $t_\alpha = \alpha$.*

Proof. Let us consider a job $i \in V_{k^\star} \backslash M$ whose first operation is processed at time $t_\alpha - 2$ for $\alpha \in \{1,\ldots,k\}$, and let $j \in M$ whose first operation is processed at time $t_\alpha - 1$. From the Sorting Algorithm, we have that $\tau_i \geq \tau_j$ and $j = i + 2$. Therefore,

$$\begin{aligned} r_i &= \tau_i + \left\lceil \frac{i}{2} \right\rceil \geq r_j - 1 \\ &\geq r^\star - 1. \end{aligned}$$

Since the second operation of job i is not available at time $r^\star - 1$, because of the idle time slot, we get $r_i \geq r^\star$ and $i \in M$, thus a contradiction. □

Lemma 10. *If $k^\star = 1$, then there exists an instance I_2 of the second special case such that*

$$\frac{\omega_A(I)}{\omega_{opt}(I)} \leq \frac{\omega_A(I_2)}{\omega_{opt}(I_2)}.$$

Proof. From Lemma 9 the Sorting Algorithm generates a schedule in which the jobs in $M = \{1, 3, \ldots, 2k-1\}$ are first processed on machine 1. Then for any job $i \in M$, we have that $\tau_i \geq r^\star - \lceil \frac{i}{2} \rceil$.

An instance I_2 of the second special case is built from I by setting $n' = 2k-1$ and $\tau'_i = r^\star - \lceil \frac{i+1}{2} \rceil$ for $i \in \{1, \ldots, n'\}$.

We first show that $\omega_A(I) = \omega_A(I_2)$. Indeed, from Lemma 8, we have that $\omega_A(I) = r^\star + |M| = r^\star + k$. Now, when considering only jobs of I_2, we have that $r_i = r^\star$. As these jobs are scheduled by the Sorting Algorithm, it follows that $\omega_A(I_2) = r^\star + k$. Thus the result.

Next, we show that, for any $i \in \{1, \ldots, n'\}$, we have that $\tau'_i \leq \tau_i$. Indeed,

1. The release date r_i for $i \in M$ verifies $r_i \geq r^\star$. So, for any $i \in M$, $\tau'_i \leq \tau_i$.
2. Every job $j \in \{1, \ldots, n'\} \backslash M$ verifies $\tau_{j+1} \leq \tau_j$ and $\tau'_j = \tau'_{j+1}$. Since $j+1 \in M$, we get $\tau'_{j+1} \leq \tau_{j+1}$. We then conclude that $\tau'_j = \tau'_{j+1} \leq \tau_{j+1} \leq \tau_j$.

Therefore, we obtain that $\omega_A(I) = \omega_A(I_2)$ and $\omega_{opt}(I) \geq \omega_{opt}(I_2)$. Thus the result. □

Lemma 11. *If $k^\star = 2$, then there exists an instance I_1 of the first special case such that*

$$\frac{\omega_A(I)}{\omega_{opt}(I)} \leq \frac{\omega_A(I_1)}{\omega_{opt}(I_1)}.$$

Proof. From Lemma 9 the Sorting Algorithm generates a schedule in which the jobs in $M = \{2, 4, \ldots, 2k\}$ are first processed on machine 2. Then, for any job $i \in M$, we have that $\tau_i \geq r^\star - \lceil \frac{i}{2} \rceil$.

An instance I_1 of the first special case is built from I by setting $n' = 2k$ and $\tau'_i = r^\star - \lceil \frac{i}{2} \rceil$ for $i \in \{1, \cdots, n'\}$.

We first show that $\omega_A(I) = \omega_A(I_1)$. Indeed, from Lemma 8, we have that $\omega_A(I) = r^\star + |M| = r^\star + k$. Now, when considering only jobs of I_1, we have that $r_i = r^\star$. As these jobs are scheduled by the Sorting Algorithm, it follows that $\omega_A(I_2) = r^\star + k$. Thus the result.

Next, we show that, for any $i \in \{1, \ldots, n'\}$, we have that $\tau'_i \leq \tau_i$. Indeed,

1. The release date r_i for $i \in M$ verifies $r_i \geq r^\star$. So, for any $i \in M$, $\tau'_i \leq \tau_i$.
2. Every job $j \in \{1, \ldots, n'\} \backslash M$ verifies $\tau_j \geq \tau_{j+1}$. But $\tau_{j+1} \geq \tau'_{j+1} = \tau'_j$. Therefore, $\tau_j \geq \tau'_j$.

Therefore, we obtain that $\omega_A(I) = \omega_A(I_1)$ and $\omega_{opt}(I) \geq \omega_{opt}(I_1)$. Thus the result. □

From Theorems 2 and 3, and Lemmas 10 and 11, we deduce the following main result.

Theorem 4. *The performance ratio of the Sorting Algorithm is asymptotically bounded by $\frac{5}{4}$. Moreover, this bound is tight.*

5 Conclusion

In this paper, we considered the unit execution time (UET) two-machine open-shop with integral time delays considerations so as to minimise the makespan. We presented a simple approximation algorithm with an asymptotic performance guarantee of $\frac{5}{4}$. We also showed that, when the time delays are all distinct, then the Sorting Algorithm produces an optimal schedule. A further research would be to exhibit other well solvable cases and heuristics with better worst case bounds.

Acknowledgment

This research is partially funded for Djamal Rebaine by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] T. Gonzalez (2005), Open shop scheduling, in J.Y.T. Leung (ed): *Handbook of scheduling*, Chapman & Hall/CRC, chapter 6, 1 – 14.

[2] W. Yu, H. Hoogeveen, J.K. Lenstra (2004), Minimizing makespan in a two-machine flow with delays and unit-time operations is NP-hard, *Journal of Scheduling*, 333 – 348.

[3] P. Lopez and F. Roubellat (2001), *Ordonnancement de la production*, Chapter 11, Hermes Science Publications.

[4] V.J. Rayward-Smith, D. Rebaine (1992), Open shop scheduling with delays, *Theoretical Informatics and Application*, 439 – 448.

[5] D. Rebaine, V.A. Strusevich (1999), Two-machine open shop scheduling with special transportation times, *Journal of the Operations Research Society* **50**(7), 756 – 764.

[6] V.A. Strusevich (1999), A heuristic for the two-machine open shop scheduling problem with transportation times, *Discrete Applied Mathematics* **2-3**, 287 – 304.

A Framework for School Timetabling Problem

Kimmo Nurmi, Jari Kyngäs
Satakunta University of Applied Sciences, Finland, Tiedepuisto 3, FIN-28600 Pori
{kimmo.nurmi, jari.kyngas}@samk.fi

Abstract: This paper introduces a framework for a highly constrained school timetabling problem, which was modeled from the requirements of various Finnish school levels. We present a successful algorithm to solve real-world problems as well as artificial test problems. Moreover, we find the best configuration for this algorithm using brute force and statistical analyses. Finally, we propose a set of benchmark problems that we hope the researchers of the timetabling problems would adopt.

Keywords: Evolutionary Algorithms, Heuristic Search, Real World Scheduling, Timetabling.

1. Introduction

In recent years many solution approaches for different timetabling problems have been introduced. Most of the work has concentrated on examination timetabling, university timetabling and course timetabling [1,2,3,4,5,6]. Timetabling researchers have obtained very promising and practicable results in these problem areas. However, school timetabling has not been as extensively studied, and widely usable results are not yet available. The school timetabling problem [7] consists of assigning lectures to periods in such a way that no teacher, class or room is involved in more than one lecture at a time and other constraints (hard and soft) are satisfied.

The main focus of this paper is to introduce a successful algorithm to tackle highly constrained school timetabling problems [8,9]. The algorithm is controlled with nine different parameters. For seven of these parameters we searched the best values using both brute force and statistical analyses. Two different methods were mainly used because we wanted to confirm that the chosen parameter values were the best ones. Two parameters were excluded because they have a great impact on the running time and therefore make the results incomparable (see Chapters 2-4).

Another point of this paper is to propose a set of benchmark instances and publish them as well as our results on the web. We wanted to lay out the foundation of comparable results (see Chapter 5). This idea is presented (by Schaerf and Di Gaspero) in [10].

2. Problem Description

We describe a school timetabling problem that arises in various Finnish school levels: secondary school, high school and college. This problem description is representative of many timetabling scenarios within the area of school timetabling. The timetabling is based on the following conditions:

- The timetable frame consists of n weeks; each week has m days and each day t periods (timeslots).
- The lecture is a predefined combination of a teacher, a class (or course), a room and the length of the lecture in periods.
- The set of teachers, classes/courses and rooms is fixed.
- More than one teacher can teach a particular class at the same time.
- Each class should have at least a given number of periods in a day, but should not have more than another given number of periods.
- The classes/courses can have common students.
- Some classes/courses must precede other classes/courses
- The rooms can be classified to certain room types.

The school timetabling problem consists of scheduling the predefined lectures in such a way that the solution is feasible and mostly acceptable to both school staff and teachers. A feasible solution satisfies the following *hard constraints*:

1. No teacher, class/course or room is scheduled to the same period more than once (basic model).
2. No class/course is scheduled to the same period, if they have common students.
3. The given lectures are scheduled to the same period.
4. The given lectures are scheduled for predetermined periods.
5. For each class the daily minimum and maximum number of periods is respected.
6. A class/course c_1 is scheduled to earlier in the week than class/course c_2, if c_1 must precede c_2.
7. A lecture cannot be scheduled to periods where the teacher, the class or the room is unavailable.

A solution is most acceptable if it satisfies the following *soft constraints*:

1. The timetable of each class should have as few idle (leap) periods as possible.
2. The school day of some classes should not start in the first period.
3. The school day of some classes should end as early as possible.
4. The timetable of some teachers should have as few idle periods as possible.
5. For some teachers the preferred daily minimum and maximum number of periods is given.
6. Some teachers prefer not to be scheduled in certain time periods.
7. Some teachers should be scheduled only on a limited number of days.
8. The lessons of a subject should be on different days.

The above formulation covers school timetabling problems occurring in Finnish secondary schools, high schools and colleges. In lower school levels the problem reminds the basic school timetabling problem and in higher levels it is more similar to a combination of the school timetabling problem and the course timetabling problem [11].

3. The Algorithm

The algorithm presented here is an extension of the *h-HCGA* algorithm presented in [12]. We have spent "quiet timetabling research life" since the development of the algorithm and returned to the problem only recently, when we were asked to schedule the timetables of one secondary school and one high school. First we spent quite a lot of time trying to improve our original algorithm with the ideas of Ant Algorithms [13, 14]. Unfortunately we were not able to find competing results. We moved into improving the parameter values of the *h-HCGA* algorithm. Table 1 summarizes the parameters and the values from which we were searching the best configuration. Because two of the parameters have a great effect on computational time, we fixed their values in order to keep the results comparable.

Extended h-HCGA parameters	
F. Population size	20 *(fixed)*
1. Reproduction/selection	Random-average, Random-best, Marriage-best
F. Maximum move sequence	10 *(fixed)*
2. Tabulist size for the GHCM-operator	0, 5, 10
3. Tabulist size for overall moves	0x, 1.5x, 2x, 4x (maximum move sequence)
4. Type of removal check	All, Hard, Hard or Soft
5. Fitness calculation	Absolute, Weighted
6. Update frequency of hard constraint weights	5, 10, 20
7. Prevention of same lessons	No, Yes

Table 1: Parameter summary of the extended h-HCGA algorithm.

The algorithm is a genetic algorithm [15,16] with one mutation operator and no recombination operators. The two most important features of the algorithm are *the greedy hill-climbing mutation (GHCM) operator*, which generates a new solution candidate from the current solution and *the*

adaptive genetic penalty method (ADAGEN), which is a multiobjective optimization method [17,18,19,20].

The GHCM operator is based on moving a lecture l_1 from its old period p_1 to a new period p_2, moving another lecture l_2 from period p_2 to a new period p_3 and so on, ending up with a sequence of moves. The initial lecture selection is random. The new period for the lecture is selected considering all possible periods and selecting the one which causes the least increase in the cost function when considering the relocation cost only. Moreover, the new lecture from that period is again selected considering all the lectures in that period and picking the one for which the removal causes the most decrease in the cost function when considering the removal cost only. The operator stops if the last move causes an increase in the cost function value and if the value is larger than that of the previous non-improving move.

We noted in [12] that we can improve the GHCM operator by introducing a tabu list, which prevents reverse order moves in the same sequence of moves. We can also prevent only some of the moves (parameter 2). To further widen the use of the tabu list, we can prevent reverse order moves in consecutive move sequences (parameter 3), that is in consecutive applications of the GHCM operator.

The ADAGEN method is an adaptive penalty method for multiobjective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints. The weights are updated in every k^{th} (parameter 6) generation using a somewhat complicated formula [12].

The reproduction operation [21] of the (genetic) algorithm is based to a certain extent on the steady-state reproduction [22]. We select randomly a timetable from the population of timetables for single GHCM operation. The new timetable replaces the old one if it has better or equal fitness. We have three variations (parameter 1). In the first one the new timetable replaces also the least fit in the current population if it is better than the current population average. In the second one the least fit is replaced with the best one, when n better timetables have been found, where n is the size of the population. In the third one we use the second variation but select a timetable using a marriage selection [23].

When selecting a lecture to be removed from a period (parameter 4), we can consider either all the lectures in that period or only those that have at least one constraint violation with the lecture that was moved to that period previously. We can also consider only hard constraint violations. When calculating the best lecture to be removed and the best period to move to (parameter 5), we can use either absolute number of constraint violations or the weighted penalty value of the ADAGEN method.

We also tried (parameter 7) to programmatically prevent lessons of a subject to be on the same day (soft constraint 8). Another possibility would have been to change it to a hard constraint.

In preliminary experiments the extended h-HCGA algorithm found encouraging results. Some parameter values seemed to produce better results than others. Our next goal was to find the best configuration for the parameter values.

4. Finding the Configuration

We had two possible choices for the process of finding out the best configuration of parameters: brute force and statistical methods. The brute force approach was very attempting because we had the possibility to use up to 78 computers. We decided to use brute force but also analyze the runs using standard statistical methods. We also performed "what-if" analyses on the runs using the

Friedman test [24] in similar fashion to Birattari & al. [25]. We used the highly constrained benchmark instance C3 (see chapter 5) in our test runs.

For the runs to be comparable in computing time we had to fix two parameter values (see table 1). The rest of the parameter values were such that they did not influence the convergence time. The total number of configurations was 1296 but because the *tabulist size for overall moves* has no effect when *tabulist size for GHCM-operator* is zero, we were left with 972 configurations.

Our goal was to find a configuration which would minimize the hard constraints to zero. The configurations were ranked according to hard constraints. We decided to run every configuration five times and for two hours each, on an AMD Athlon 1700+ with 512Mb of memory. For every generation of the algorithm we saved the number of hard constraint violations.

At first we only considered those runs that had ended up with zero hard constraint violations ─ 218 out of 972 runs met this criterion.

4.1. Choosing the Configuration

We started analyzing the results by counting how many times each value of each parameter occurred in the best runs. Table 2 summarizes the results.

Parameter	Values and occurrences
1. Reproduction/selection	**Random-average = 38%**, **Random-best = 40%**, Marriage-best = 22%
2. Tabulist size for GHCM-operator	0 = 20%, **5 = 42%**, **10 = 38%**
3. Tabulist size for overall moves	**0x = 18%**, **1.5x = 25%**, **2x = 26%**, **4x = 31%**
4. Type of removal check	All = 29%, **Hard = 63%**, Hard or soft = 8%
5. Fitness calculation	**Absolute = 64%**, Weighted = 36%
6. Update frequency of hard constraint weights	**5 = 48%**, 10 = 30%, 20 = 22%
7. Prevention of same lessons	**No = 77%**, Yes = 23%

Table 2: The chosen values (bolded) after brute force runs.

The best value for the last four parameters could easily be chosen (tested with the t-test). The first three, on the other hand, required further investigation.

Next we ran the chi-squared test [24] pairwise between all parameters. We were especially interested in those test cases which included the chosen parameter values. Unfortunately none of the chi-squared tests showed statistically significant dependencies among the parameter values. We enlarged the chi-squared test to include also those runs that had ended up with one hard constraint violation, but that did not change the test results. (These statistical tests can be obtained from us at request.)

At this phase we were still not able to find the values for the first three parameters. However, we were able to reduce the number of free parameters from 972 to 16 (the combination of chosen parameter values of each parameter). For the *reproduction/selection*, the *tabulist size for GHCM-operator* and the *tabulist size for overall moves* we were not able to choose one specific value. That strengthened our decision to use statistical methods.

4.2. Statistical Methods

Birattari & al. [25] have developed a process, F-Race, in which they start with a considerable large population and eliminate individuals from it with the help of statistical methods. The idea is to eliminate an individual from the population as soon as it can be statistically significantly proven that the individual is not performing good enough. The process can be thought of as a race where

every individual competes against each other. At specific intervals the performance of each individual is checked and the poorly performing ones are dropped out from the population. This process is well suited for problems with extensive number of parameters.

We tested this method to see how it would have worked in this case. However, we did not use the actual racing algorithm ─ we only tested the runs to see which ones would have dropped out after two hours of running. From the output of the runs we sampled ten evenly spaced observations. This meant that we had 972 treatments (configurations) and 10 blocks (samples) for the Friedman test.

Table 3 summarizes the results of the Friedman test for the case of the eliminated runs. When we compare this with table 2 we can see that these results are very compatible with each other. For example, table 2 tells us that in the 218 runs 63% of the configurations had parameter *type of removal check* set to value *Hard*. Table 3, on the other hand, tells us that only 7% of the eliminated runs included configurations where this parameter was set to value *Hard*. The information in the two tables supports each other very well.

Parameter	Values and occurrences
1. Reproduction/selection	**Random-average = 21%**, Random-best = 34%, Marriage-best = 45%
2. Tabulist size for GHCM-operator	0 = 45%, **5 = 24%**, 10 = 31%
3. Tabulist size for overall moves	0x = 29%, 1.5x = 20%, 2x = 28%, 4x = 23%
4. Type of removal check	All = 21%, Hard = 7%, Hard or soft = 72%
5. Fitness calculation	Absolute = 14%, Weighted = 86%
6. Update frequency of hard constraint weights	5 = 16%, 10 = 28%, 20 = 56%
7. Prevention of same lessons	No = 30%, Yes = 70%

Table 3: The chosen values (bolded) in eliminated configurations.

In the brute force runs we were not able to choose the best values for the first three parameters. Here the best value for the *reproduction/selection* is *Random-average*. The difference between the values *Random-average* and *Random-best* is statistically significant at 0.001 level when measured with the t-test. Therefore we chose the value *Random-average* for parameter *reproduction/selection.*

The other two parameters show no statistical difference between the values. We chose value *5* for the parameter *tabulist size for GHCM-operator* because both tables indicate that this could be the best value (though not statistically significantly differing from value *10*). The parameter *tabulist size for overall moves* shows no difference in goodness of the values. The value was chosen to be *2* because we *felt* it could be the best value.

5. Standard Benchmark Instances

In school timetabling we do not have a set of standard test problems, as is the case in examination timetabling [10]. We now introduce some test instances and hope that they will lay the foundation for the standard universal benchmark instances for school timetabling problems.

The first set of test problems consists of *all-N-problems* introduced in [12]. These artificial problems have a timeframe of N days with N periods in one day totaling N^2 periods. There are N teachers, N classes and N rooms. All possible combinations of a teacher, a class and a room are to be scheduled. This will give us N^3 lectures to be scheduled. The problem is quite difficult to solve since a feasible solution has no idle periods for any teacher, class or room.

The second set of test problems consists of *Abramson-N-problems* introduced in [26]. These artificial problems have a timeframe of 30 periods. The lectures are built by first choosing a random

teacher and class and room identifiers and then placing that lecture in the first possible period. If the lecture cannot be placed in an available period, then it is discarded and another one is generated. This process continues until all teacher, class and room identifiers have been used. The resulting timetable has again no idle periods and thus is tightly constrained.

Both *all-N-problems* and *Abramson-N-problems* are excellent test instances since

a) every single school timetabling algorithm can tackle them because they are the simplest version of the problem (basic model) and
b) we can easily increase the problem difficulty by increasing *N*.

Problem identifier	B3	S3	H3	C3
School level	Artificial school	Secondary school	High school	College
#Weeks	1	1	1	1
#Days	5	5	5	5
#Periods per day	4	7	7	8
#Teachers	21	25	18	46
#Classes/courses	11	14	50	41
#Rooms/room classes	3	25	13	34
#Lectures	169	280	219	387
#Periods in lectures	200	306	320	854
#Clashes (hard constraint 1)	3020	8590	6420	28574
#Overlapping classes/courses (hard constraint 2)	11	0	52	20
#Unavailabilities (hard constraint 7)	108	600	72	328
Soft constraint 1 in use	Yes	Yes	No	Yes
Soft constraint 4 in use	Yes	Yes	Yes	Yes
Soft constraint 6 in use	No	Yes	Yes	No
Soft constraint 8 in use	Yes	Yes	Yes	Yes

Table 5: Summary of one artificial and three real-world problems.

The third set of problems consists of one artificial problem and three small to medium-sized real-world problems in Finnish schools, one from each school level. Table 5 summarizes these four problems. The data for these problems has been published on the web [27]. In all of these problems we are searching for a feasible solution which optimizes soft constraints 1, 4, 6 and 8. Other soft constraints are not in use. These four problems are good test instances since

a) most of the school timetabling algorithms should be able to tackle them because they are not too complicated versions of the problem,
b) they are quite moderate-sized and
c) they are different from each other.

The C3 problem is the most constrained and thus the most interesting one. We solved six problems from the standard benchmark instances: All-11-problem, Abramson-15-problem, B3, S3, H3 and C3. We used the parameter values found in the statistical analysis. To keep the results fair we did no further fine-tuning to the extended h-HCGA algorithm. The test runs were performed using the same computers as in Chapter 4. Table 6 summarizes the results.

Furthermore, we decided to run the algorithm using a simulated annealing refinement introduced in [12]. When the GHCM operator selects a new period for a lecture l_i, we apply the following selection mechanism. Periods are examined in random order. A period is selected as the current best one if its fitness is less than the current best one or if the current annealing temperature [28] ac-

cepts their difference. The same mechanism is used in the GHCM operator while removing a lecture. Note that the GHCM operator does not accept upward (increasing cost) move sequences, even if a single move can be upward. Table 6 summarizes the results using the simulated annealing refinement. It also shows the best manual solution the staff of the schools were able to find. The best solutions we have found in all of our experiments can be found in [27].

Problem identifier	Nbr of runs	Single run time	Best	Median	Worst	Best manual
Extended algorithm						
All-11	8	8 hours	0-0	0-0	0-0	
Abramson-15	8	8	2-0	3-1	4-0	
B3	8	8	0-3	1-0	1-6	
S3	8	8	0-2	0-2	0-3	
H3	8	8	0-3	0-3	0-3	
C3	8	8	0-2	0-4	0-6	
With SA refinement						
All-11	8	24 hours	0-0	0-0	0-0	
Abramson-15			2-0	2-0	3-1	
B3			0-2	0-5	1-1	
S3			0-2	0-2	0-2	0-8
H3			0-3	0-3	0-3	0-12
C3			0-1	0-1	0-3	0-10

Table 6: Results for standard benchmark instances and the best manual solutions. For example, the value 1-6 stands for one hard constraint and six soft constraint violations.

6. Conclusions and further work

In this paper we considered a highly constrained school timetabling problem and presented a successful algorithm to solve real-world problems as well as artificial test problems. Our algorithm performs very well in the timetabling problems presented in this paper. In order to find out if statistical methods could be of any help in solving the parameter values we ran our algorithm with all possible configurations and analyzed the results using the Friedman test. The Friedman test clearly implied that it is able to extract poor configurations in favor of the good ones. In the near future, we will build the racing process into our program. Our research has shown that the chosen parameter values were very good for the problem C3, but we do not know if they are the best for all problems. Therefore it is best to start with as many configurations as possible and reduce them in time. We have developed a web page [27] that allows other researchers to download the problem description, the standard benchmark instances and the computational results.

References

[1] E.K. Burke and P. De Causmaecker (2003). The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th International conference, Gent 2002, Springer Lecture Notes in Computer Science, vol. 2740, Springer.

[2] E.K. Burke and M. Trick (2005). The Practice and Theory of Automated Timetabling V: Revised Selected Papers from the 5th International conference, Pittsburgh 2004, Springer Lecture Notes in Computer Science, vol. 3616, Springer.

[3] E.K. Burke and Hana Rudová (2006). Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Masaryk University.

[4] A. Schaerf (1999), A Survey of Automated Timetabling, Artificial Intelligence Review **13**(2), 87 – 127.

[5] E.K. Burke and S. Petrovic (2002). Recent Research Directions in Automated Timetabling, European Journal of Operational Research **140**(2), 266 – 280.

[6] B. McCollum (2006). University Timetabling: Bridging the Gap between Research and Practice. In *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, 15 – 35.
[7] D de Werra D (1985). An Introduction to Timetabling. European Journal of Operations Research **19**, 151–162.
[8] S. Even, A. Itai, and A. Shamir (1976). On the complexity of timetable and multicommodity flow problems. SIAM Journal on Computing **5**, 691 – 703.
[9] T.B. Cooper and J.H. Kingston (1996). The Complexity of Timetable Construction Problems, in the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, Springer-Verlag (Lecture Notes in Computer Science), 283 – 295.
[10] A. Schaerf and L. Di Gaspero (2006). Measurability and Reproducibility in Timetabling Research: State-of-the-Art and Discussion. In *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, 53 – 62.
[11] A. Schaerf (1999). A Survey of Automated Timetabling. Artificial Intelligence Review **13**(2), 87 – 127.
[12] K. Nurmi (1998). Genetic Algorithms for Timetabling and Traveling Salesman Problems. *Ph.D. dissertation*, University of Turku, Finland.
[13] M. Dorigo, V. Maniezzo, and A. Colorni (1996). The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, **26**(1), 29 – 41.
[14] M. Dorigo, G. Di Caro, and L. M. Gambardella (1999). Ant Algorithms for Discrete Optimization. Artificial Life, **5**(2), 137 – 172.
[15] Goldberg, David E (1989), Genetic Algorithms in Search, Optimization and Machine Learning, Kluwer Academic Publishers, Boston, MA.
[16] Vose, Michael D (1999). The Simple Genetic Algorithm: Foundations and Theory, MIT Press, Cambridge, MA.
[17] J.D. Landa Silva, E.K. Burke and S. Petrovic (2004). An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling, MetaHeuristics for Multiobjective Optimisation (edited by X.Gandibleux, M.Sevaux, K.Sorensen and V.T'Kindt), Springer Lecture Notes in Economics and Mathematical Systems Vol. 535, 91 – 129.
[18] Fonseca C.M, Fleming P.J (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation **3**(1), 1 – 16.
[19] Smith A.E, Tate D.M (1993). Genetic Optimisation using a Penalty Function. In *Proceedings of the 5th International Conference on Genetic Algorithms*, 499 – 503.
[20] Richardson J.T, Palmer M.R, Liepins G, Hilliard M (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 191 – 197.
[21] D Goldberg (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.
[22] G Syswerda (1989). Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 2 – 9.
[23] P. Ross and G.H Ballinger (1993) PGA - Parallel Genetic Algorithm Testbed. Department of Articial Intelligence. University of Edinburgh.
[24] W.J. Conover (1999), *Practical Nonparametric Statistics*, John Wiley & Sons, New York.
[25] M. Birattari & al. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 11– 18.
[26] D. Abramson, M. Krishnamoorthy M, H. Dang (1999). Simulated Annealing Cooling Schedules for the School Timetabling Problem. Asia-Pacific Journal of Operational Research **16**, 1 – 22.
[27] K. Nurmi, J.Kyngäs (2007). School Timetabling Problem – Formulation, Instances and Computional Results. URL: www.samk.fi/sttp.
[28] P.J.M van Laarhoven, E.H.L Aarts E.H.L (1987). Simulated annealing: Theory and applications. Kluwer Academic Publishers.

A Memetic Algorithm for Solving a Timetabling Problem: An Incremental Strategy

Ender Özcan, Alpay Alkan
Yeditepe University, Computer Engineering Department, İstanbul, Turkey,
{eozcan, aalkan }@cse.yeditepe.edu.tr

Timetabling problems are well known complicated constraint satisfaction problems. A new real-world course timetabling problem is described in this study. The difficulty in approximating to an optimal solution in a reasonable time increases for the large problem instances regardless of the algorithm used. An incremental strategy that aims to relieve this difficulty to some extent combined with a memetic algorithm (MA) is investigated for solving the new timetabling problem. The incremental MA is a multistage approach that enlarges the size of the candidate solutions by adding a selected subset of unscheduled course meetings at a stage. A solution is then sought for all the course meetings at hand. The initial results show that the incremental MA is promising.

Keywords: Memetic Algorithms, Timetabling, Multistage Approach, Hill Climbing.

1. Introduction

The high-school graduates in Turkey enter a country-wide *Student Selection Examination* (OSS), in order to get admitted to a higher education program at a university. The *Student Selection and Placement Center* (OSYM) has been providing a centralized system for fair access and placement of students to higher education programs since 1974. Currently, OSYM administers OSS. The students are placed into the higher education programs according to their OSS score and set of choices. OSS is arranged every year. Less than 40% of the applicants can continue their education in an undergraduate degree program. This situation causes a high-level competition among students. For this reason, there are many private OSS preparation schools in Turkey established as a support mechanism for the candidates. Such schools opening more branches started to deal with the difficult problem of scheduling the course meetings properly subject to a set of constraints.

The timetabling problems are NP complete [11] real-world problems. A comprehensive survey on timetabling can be found in [7] particularly discussing the MAs. Course timetabling problems (CTPs) are a subset of timetabling problems that require an optimal arrangement of resources for the employees (e.g., teachers), employers (e.g., school administration) and students at an educational institution subject to a set of constraints. There are two types of constraints in timetabling: *hard* and *soft*. Hard constraint violations represent highly undesirable situations, while soft constraints are less essential preferences. The constraints can be categorized further for the practical timetabling. In this paper, a new course timetabling problem is introduced, University Exam (OSS) Preparation School Timetabling Problem (PSTP). PSTP is analyzed initially by Alpay Alkan during his studies on timetabling.

Alkan and Ozcan introduced a violation directed hierarchical hill climbing method (VDHC) in [3] for solving the university course timetabling problem. VDHC performs a local search over the *constraint oriented neighborhoods* as described in [29] based on the constraint types, their violations and structure of the problem. Ozcan extended the study and suggested a heuristic template for designing a set of operators in [22]. A variety of genetic operators and self-adjusting hill-climbers based on this template have already been experimented within the Memetic Algorithm (MAs) for solving different timetabling problems in [21] and [23]. The VDHC methods within the MAs that managed multiple constraint based hill climbers turned out to be successful in timetabling. In this study, an incremental MA is proposed for solving PSTPs. The performance of the incremental MA is compared to the conventional one both using the VDHC on real PSTP instances, each requiring a large set of course meetings to be scheduled.

2. Preliminaries

2.1. High-school Course Timetabling Problem

The university and high-school course timetabling problems are the subsets of CTPs that are commonly dealt with by numerous researchers. There is a variety of approaches used for solving such problems that have different types of constraints. In general, the high-school course timetabling differs from the university course timetabling. Besides the set of constraints, the timetable for a student at a high-school has less empty time-slots as compared to a student at a university. PSTPs form another subset of CTPs that are more similar to the high-school timetabling problems.

There are exact and inexact approaches for solving the high-school timetabling problems. The researchers focus on the inexact methods more, including heuristics, meta-heuristics and *hyper-heuristics* [5], [19]. Hyper-heuristics perform a search over a set of heuristics. Most of the existing hyper-heuristic combine two decision making strategies. A heuristic selection mechanism is used for choosing a heuristic from a lower set of heuristics, while an acceptance mechanism is used for deciding whether to accept or reject a candidate solution [4]. Different meta-heuristics for high-school course timetabling have been studied by many researchers, such as simulated annealing [1], [2], evolutionary algorithms [10], [13], [25], tabu search [16], [28]. Colorni et. al. [9] compares these approaches over an Italian high-school data.

2.2. University Exam Preparation School Timetabling Problem (PSTP)

OSS is a single stage exam having two parts, in which two aptitudes of the entering candidates are assessed: verbal and quantitative. Computing the score of a candidate's exam requires some transformations of the raw result by taking into account the grade-point average of the student at school, the difficulty level of each question that is determined statistically, etc. After the transformations are applied, three different composite score types can be computed: *verbal*, *quantitative* and *equally weighted* OSS scores. One of these scores is used in the selection of those candidates who will be considered for the placement to the undergraduate programs. Each department at a Turkish university admits students according to a specific OSS score type. For example, a computer engineering department accepts students based on their quantitative OSS scores. Hence, the students aim to maximize one of those OSS score types to get admitted to a department according to their wish at a university by correctly answering the relevant questions. The private preparation schools (PPSs) act as a support mechanism for the students that will enter the OSS. A student can be admitted to a PPS at any time during his/her high-school education. The high-school education has been extended from 3 to 4 years in Turkey.

In a PPS that has several branches at different locations, there are teachers that circulate around these branches, where each registered student attends a set of courses at a specific branch. A student gets prepared to maximize one of the scores, verbal, quantitative or equally-weighted. A *division* indicates the score type that a student aims to collect during OSS. The curriculum of a verbal second-*grade* (*year*) high-school student in a PPS differs from that of a verbal or quantitative third-grade high-school student. Hence, the set of courses offered for each student differs according to his/her division and *grade*. At a branch, depending on the number of registered students, several sections might be arranged to cover them all. Depending on the high-school, the classes might be held before noon, in the afternoon or during the whole day. Moreover, some OSS applicants are high-school graduates. Consequently, PPSs have to arrange course meetings accordingly. For example, eight different sections might be required for all quantitative second-grade high-school students. Some of these sections might require the courses to be scheduled in the morning and some of them in the afternoon during the weekdays. So, all grades are further divided into grade sections. The third-year high-school students that are in the quantitative division must take mathematics, natural sciences and Turkish language courses. All the students that are in the verbal or equally weighted divisions must take some courses in social sciences such as geography and history as well as mathematics and Turkish language courses. But, the number and length of the meetings that must be assigned to the grade sections (of different divisions) can differ. For example, a student in a grade section of an equally weighted division must attend four meetings of the

geography course, whereas a student in a grade section of a verbal division must attend six meetings of the geography course. Each grade section groups a set of course sections that a student must attend. A course section denotes a set of meetings for a course.

In a University Exam Preparation School Timetabling Problem (PSTP), an optimal assignment of all events that places course-section meetings into p periods in a timetable of d days by t timeslots per day is searched. This assignment is subject to a set of constraints. If the total number of course meetings is M, then the search space size becomes M^p. The traditional approaches might fail to obtain an optimal solution. An assignment in a PSTP is an ordered pair (x,y), where $x \in E$ (set of course-section meetings) and $y \in T$ (domain, set of periods). The interpretation of this assignment in terms of PSTP is: "Course section meeting x starts at time y". Most of the teachers in a PPS are part time teachers having preferences that are treated as hard constraints. There might be inexperienced teachers who are organized to enter some courses together with the experienced teachers. Each teacher is assigned to a course section beforehand and can take responsibility for more than one course-section. Some of these course sections might be held in different branches. Hard constraints are as follows:

C1. Each meeting of a course section should be assigned to a period in a different day
C2. Meetings in a grade section cannot overlap
C3. Meetings of a teacher cannot overlap
C4. A teacher can prefer certain days and/or periods
C5. The administration can prefer certain days and/or periods for some course meetings
C6. Each grade section excludes some periods from before or after noon
C7. Some course meetings can be scheduled at the same time for some grade sections
C8. More than one teacher can be assigned to a section of a course

Soft constraints are as follows:

C9. There shouldn't be an interleave of more than some given number periods between the meetings of a teacher in each day so that a teacher would not wait too long for its next meeting
C10. There is a minimum and a maximum load per day for teachers

Solving the PSTP requires the generation of a *feasible* timetable containing a collection of assignments one per course section meeting with the minimum number of constraint violations. A few number of C5, C7 and C8 constraints appear among all constraints in a problem.

3. An Incremental Strategy and Memetic Algorithms for PSTPs

Memetic Algorithms (MAs) represent a set of hybrid algorithms that combine Genetic Algorithms (GAs), advocated by Holland [17] and local search. An *individual* (chromosome) in a GA represents a candidate solution for a given problem. Each *gene* that composes an individual receives an *allele* value from a set of values. For example, in a binary representation, a gene gets an allele value from the set {0,1}. In an iterative cycle, a randomly generated *population* of individuals evolves towards an optimal solution by undergoing a set of genetic operators, namely *crossover*, *mutation* and *selection*. In a conventional MA, a hill climbing method is applied after the mutation occurs. The GAs and MAs are successfully used in solving a set of difficult search and optimization problems, including the real-world timetabling problems ([3], [14], [18]–[25]). In most of the meta-heuristic approaches, as the problem size increases, the speed might become an issue. A *reasonable* amount of time should be spent on obtaining a *reasonable* schedule for the timetabling of large problem instances. That's why Carter uses a divide-and-conquer approach in solving such problems [8]. The traditional approaches such as integer programming are suggested for solving sufficiently small problem instances during the conquering step. Weare [30] and Burke and Newall [6] showed the potential in applying a multistage approach for solving timetabling problems.

Most of the approaches for timetabling can be converted into a multistage approach based on a strategy that somehow selects and handles a subset of events during each stage. Three different types of such strategies can be identified. In the first type, the stages can be arranged such that the approach is applied only to a selected subset of events. Once a satisfactory solution is obtained, based on some criteria only for these events subject to the constraints, the assignment for each

event is fixed. Then the next subset is processed as shown in Figure 1(a). As a second type of strategy, the approach can be applied to the union of an unprocessed subset indicating some unscheduled events and a subset of some previously processed events as illustrated in Figure 1(b). Still some scheduled events are fixed in this strategy and they are not processed further by the approach. Both of these strategies are evaluated in [30] and [6]. They do not seem to be that promising for most of the common real-world timetabling problems such as nurse rostering, or course timetabling. Whenever a solution to a subset is fixed, the approach is disallowed for exploring some part of the search landscape, which might contain a promising area. A third extreme strategy is utilized in this study. No assignment of events is fixed. At each stage, a subset of unscheduled events is incrementally added to be processed by the approach along with the previously processed subset of events as in Figure 1(c). The aim of this study is to inquire whether such an incremental multistage approach can provide a better performance as compared to its single stage version.

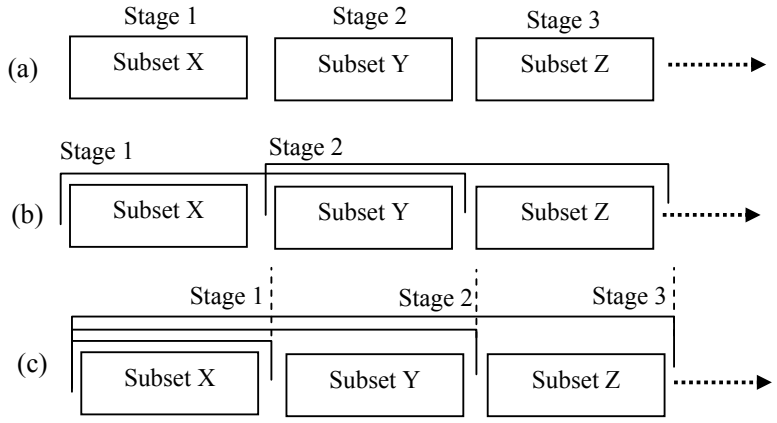


Figure 1. Possible ways to arrange stages in a multistage approach for timetabling

As an incremental multistage approach, an MA is used for solving PSTPs as illustrated in Figure 2. At each stage, a subset of new (unscheduled) events is selected using certain criteria. Unscheduled events for the selected subset of events are randomly generated within a population of candidate solutions. This population is exposed to the traditional MA operators. Whenever some stage termination criteria is satisfied, then another subset of new events is chosen. This process is repeated until all events are scheduled and some additional termination criteria is satisfied. After the first pass, each individual in the initial population is a partial solution and has a small size. At each stage, the size of individuals incrementally grows as the new subset of events is added for optimization. In this way, no portion of the search landscape is ignored.

```
Repeat
      Select a group (subset) of new events based on some criteria
      Generate random assignments for the new events in all individuals
      Repeat
            Apply Crossover, Mutation and then Hill Climbing
      Until stage termination criteria1 are satisfied
Until all events are scheduled and termination criteria2 are satisfied
```

Figure 2. Pseudo-code for an incremental Memetic Algorithm for timetabling

An individual contains all course (section) meetings to be scheduled and its physical implementation reflects the same logical arrangement in Figure 3. A gene corresponds to a course section in the representation used. All course section meetings are generated randomly without any clash for all individuals within the initial population. Similarly, the traditional mutation randomly reschedules the meetings in a course section with no clash using a probability of 1/*no_of_course_sections*. By this way, C1 is satisfied at all times. Two crossover operators are implemented forming two new individuals. Modified one-point crossover (m1PTX) swaps parts of

selected individuals at a selected point. As a crossover point, one of the start points of the grade sections is randomly chosen. A modified uniform crossover (mUX) exchanges course meetings in each grade section as a whole with a probability of 1/2.

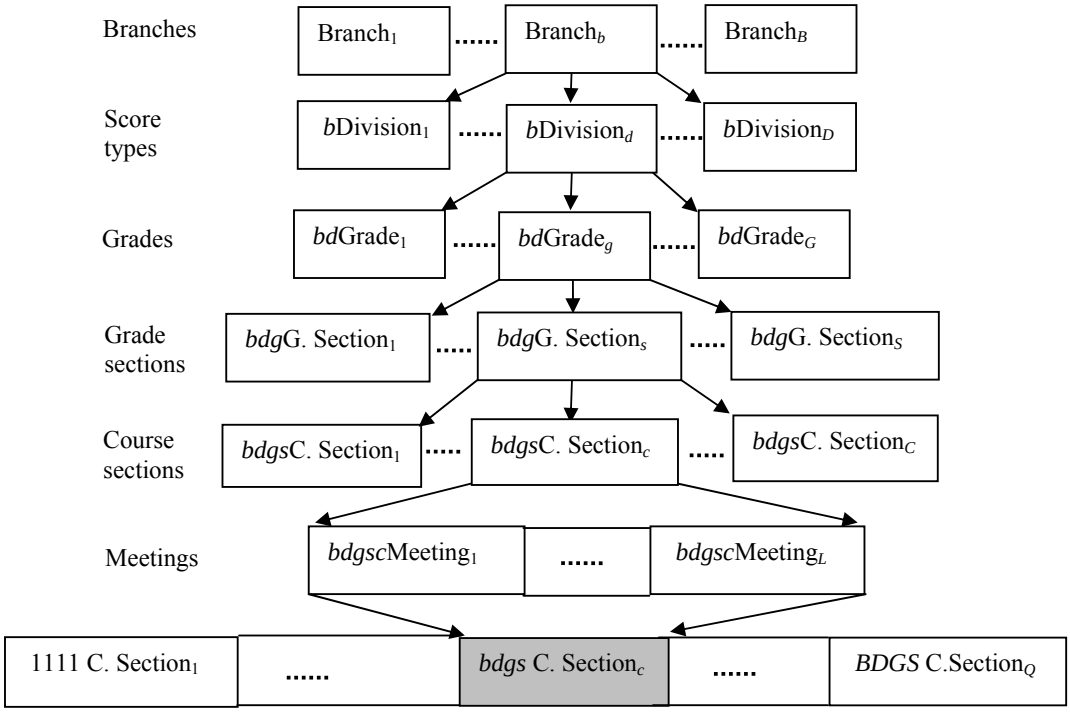


Figure 3. The logical organization of the course (section) meetings in a PSTP and the representation used in the MAs, where "$bdsg$ C. Section$_c$" denotes all course meetings of the c^{th} course section of the s^{th} grade section from the g^{th} grade of the d^{th} division in the b^{th} branch of a school

For C7, the course sections that should be scheduled to the same periods have just a single gene in the representation pointed by them. Similarly, each course section that will be taught by more than one teacher has a single gene pointed by the related teachers for C8. The relevant violations are detected by using the list of each teacher and course section assignments. No assignment, out of the restricted domain of each course meeting due to C4-C6 is allowed during the evolution. The MAs utilize a set of constraint based hill climbers for C2, C3 and C9, C10. Each hill climber attempts to remove the violations due to a constraint by random rescheduling. A limited number of possibilities are tested and the best one is accepted. The hill climber for C2 is applied after all genetic operators.

The idea of using a violation directed mutation operator in a genetic algorithm is tested over a set of real and syntactic data for lecture and examination timetabling by Ross et. al. ([26], [27]). Their results show that the random selection of a gene and then the selection of an allele to be assigned by using tournament perform the best. The tournament strategy favors the assignment that produces less number of violations. Extending this idea further, a mechanism called violation directed hierarchical hill climbing (VDHC) as described in [3] and [21] is used to manage the hill climbers for C3, C9 and C10. More details on the VDHC can be found in [22]. At each step, the number of violations due to a constraint type is computed and a hill climber is randomly selected using a tournament strategy (with a tour size of 2) based on this information. Each hill climber aims to correct the violations of the related constraint type. Then the selected hill climber is applied to a group of course meetings. The VDHC makes a hierarchical traversal over the groups of course meetings based on the static organization of events (Figure 3) until there is no improvement or a maximum number of steps is exceeded. The fitness function is a weighted sum of the number

of all constraint violations. Each conflict counts as one violation in C2 and C3, while the number of violations is the total deviation from the limits for C9 and C10. For example, if a teacher has a load of 1 in a day and the minimum load is per day is 3; this generates a violation of 2. In the incremental MA, all grade sections receive an additional selected course section simultaneously at a stage. The course section with the largest number of meetings in each grade is added. In case of equality, a random choice is made. The MA at a stage terminates whenever all hard constraints (C1-C8) are satisfied by an individual or a maximum number of steps is exceeded.

4. Experiments

Pentium IV 3 GHz. windows machines having 2 Gb of memory are used during the experiments. All runs are repeated fifty times. *Success rate* (s.r.) indicates the ratio of successful runs, achieving the expected fitness to the total number of runs. A real data obtained from *Final Dershanesi*, a private PPS is used during the experiments. This data is divided into smaller pieces and mixed to perform the initial experiments as summarized in Table 1. Even these small problem instances are difficult enough, necessitating the application of a nontraditional approach. Each problem instance requires an optimal schedule to be generated for more than 1100 course meetings. There are 8 days and 12 hours per day in the timetable. For most of the courses, one hour is required for each meeting. The students in a grade section attend 45 to 48 course meetings. Each course section requires 2 to 8 course meetings. The interleave in C9 is fixed as one. The minimum and maximum load of a teacher imposed by C10 is set to 2 and 6 hours per day, respectively. The benchmark data is available at http://cse.yeditepe.edu.tr/~eozcan/research/TTML/.

Table 1. The characteristics of the experimental data set, where *minl* and *maxl* denote the minimum and maximum total load of the teachers for a given problem, respectively

label	*No. of Meetings*	*No. of Branches*	*No. of Divisions*	*No. of Grades*	*Grade Sections*	*Course Sections*	*No. of Teachers*	*minl*	*maxl*
fd1	1107	1	3	1	30	374	41	5	46
fd2	1128	2	4	3	30	322	49	6	40
fd3	1131	2	4	3	30	322	50	8	42
fd4	1163	2	3	2	30	342	42	6	46
fd5	1166	2	3	2	30	342	42	8	41
fd6	1187	1	4	3	30	290	48	3	46

The incremental MA is compared to the conventional MA, which attempts to schedule all course meetings simultaneously. For a fair comparison between the approaches, the experiments are terminated if the execution time exceeds 600 CPU seconds or the expected global optimum is achieved. If there are no constraint violations, 0 fitness value is expected. Equal weights are used within the fitness function. Tournament strategy is used to choose individuals for crossover with a tour size of four. Crossover is applied to the individuals with a probability of 0.5. As a replacement strategy, the best two individuals in a generation are passed to the next one. The rest of the population is generated using the genetic operators. Some preliminary experiments are performed using the multistage MA. Population size of 16 and 32 are compared. The results show that a small population is a slightly better choice. It is also observed that the crossover mUX is slightly better than m1PTX on average. For this reason, mUX is chosen as the crossover and a population size of 16 is used during the further experiments. The rest of the settings are kept the same. The comparison of the MAs is presented in Table 2. The incremental approach performs better than the conventional MA in almost all cases. For all of the problem instances all constraints are left unresolved, except fd5. The success rate for the incremental MA on fd5 is 0.62, while it is 0.48 for the conventional MA. For fd2, the incremental MA achieves the same quality solution as the conventional one by visiting less number of states on average.

Table 2. Comparison of the MAs, where *viol*., *gen*. denote violations and generations, respectively

label	*Incremental MA*					*Conventional MA*				
	best	*avr. viol.*	*std.*	*avr. gen.*	*std.*	*best*	*avr. viol.*	*std.*	*avr. gen.*	*std.*
fd1	19	28.7	5.4	1323.3	5.4	39	52.1	8.4	1264.0	34.1
fd2	2	4.3	1.9	1152.0	20.7	2	5.4	3.1	1301.4	36.7
fd3	7	12.8	3.0	1165.1	28.6	12	19.1	4.2	1292.9	17.8
fd4	27	48.4	8.0	1204.0	25.6	49	69.2	9.3	1213.3	34.9
fd5	0	0.6	1.0	888.4	339.9	0	0.9	1.1	1030.8	472.8
fd6	2	9.7	3.9	1128.0	34.1	11	19.8	5.0	1285.9	21.5

5. Conclusions

A new course timetabling problem is presented in this paper: University Exam Preparation School Timetabling Problem (PSTP). The search space is extremely immense even for a *small* PSTP instance. A set of such problem instances is provided, each requiring an optimal schedule for more than 1100 course meetings subject to a set of hard and soft constraints. An incremental strategy is presented to convert a single stage approach into a multistage approach for timetabling. In order to obtain a *good* solution in a *reasonable* amount of time for a PSTP instance, a memetic algorithm based on such a strategy is proposed. For this purpose, in addition to the incremental approach, the population size is kept small within the MA. The comparison between the incremental MA and its conventional version shows that the incremental one achieves better results. Both MAs use the same operators, including the VDHC; a heuristic that decides the most appropriate hill climber to apply whenever necessary from a set of constraint based hill climbers.

In a PSTP, the search landscape is highly multimodal and immense due to the number of course meetings to be scheduled and the constraints to be satisfied. It is likely that an approach for solving this problem might get stuck at a local optimum. It seems this is what happens in case the conventional MA is utilized with a small population size. The proposed incremental strategy acts as a diversification mechanism within the MA due to the newly introduced events and their random scheduling at each stage. The search space size also grows gradually along with the problem size. This process seems to alleviate the burden of solving a large problem. At each stage, a new search starts over an enlarged space using the partial solutions from the previous stage. The proposed incremental strategy is a universal strategy that can be adapted easily by the existing approaches for timetabling and scheduling. Different mechanisms for the subset selection and the termination criteria at each stage can be investigated further.

Acknowledgement

This research is funded by TÜBİTAK (The Scientific and Technological Research Council of Turkey) under the grant number 107E027.

References

[1] D. Abramson (1991), Constructing School Timetables Using Simulated Annealing, Sequential and Parallel Algorithms. Management Science **37**(1), 98 – 113.

[2] D. Abramson, H. Dang, and M. Krisnamoorthy (1999), Simulated Annealing Cooling Schedules for the School Timetabling Problem, *Asia– Pacific J. of Op. Res.* **16**, 1 – 22.

[3] A. Alkan and E. Ozcan (2003), Memetic Algorithms for Timetabling, *Proc. of IEEE Congress on Evolutionary Computation*, 1796 – 1802.

[4] B. Bilgin, E. Ozcan, E.E. Korkmaz (2006), An Experimental Study on Hyper-Heuristics and Exam Scheduling, *Proc. of the 6th Int. Conf. on PATAT*, 123 – 140.

[5] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic and R. Qu (2007), A Graph-Based Hyper Heuristic for Educational Timetabling Problems, *EJOR* **176**(1), 177 – 192.

[6] E.K. Burke, J.P. Newall (1999), A Multistage Evolutionary Algorithm for the Timetable Problem, *IEEE Trans. on Evolutionary Computation* **3**(1), 63 – 74.
[7] E.K. Burke, J.D. Landa Silva (2004), The Design of Memetic Algorithms for Scheduling and Timetabling Problems. Krasnogor N., Hart W., Smith J. (eds.), *Recent Advances in Memetic Algorithms,* Studies in Fuzziness and Soft Computing, vol. 166, Springer, 289 – 312.
[8] M.W. Carter (1983), A Decomposition Algorithm for Practical Timetabling Problems, Dept. of Industrial Engineering, Univeristy of Toronto, Working Paper 83-06, April.
[9] A. Colorni, M. Dorigo and V. Maniezzo (1992), A genetic algorithm to solve the timetable problem, Politecnico di Milano, Italy , Tech. rep. 90-060 revised.
[10] W. Erben, J. Keppler (1995), A Genetic Algorithm Solving a Weekly Course– Timetabling Problem, *Proc. of the First Int. Conf. on the Practice and Theory of Automated Timetabling (ICPTAT)*, Napier University, Edinburgh, 21 – 32.
[11] S. Even, A. Itai, and A. Shamir (1976), On the Complexity of Timetable and Multicommodity Flow Problems, *SIAM J. Computing* **5**(4), 691 – 703.
[12] H.L. Fang (1994), *Genetic Algorithms in Timetabling and Scheduling*, PhD thesis, Department of Artificial Intelligence, University of Edinburgh, Scotland.
[13] G.R. Filho, L.A.N. Lorena (2001), Constructive Evolutionary Approach to School Timetabling. Lecture Notes in Computer Science **2037**, Springer, 130 – 139.
[14] D.E. Goldberg (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison– Wesley, Reading (MA).
[15] C. Head and S. Shaban (2007), A heuristic approach to simultaneous course/student timetabling Computers & Operations Research **34**(4), 919 – 933.
[16] A. Hertz (1992), Finding a feasible course schedule using a tabu search. *Discrete Applied Mathematics* **35**, 255 – 270.
[17] J.H. Holland (1975), *Adaptation in Natural and Artificial Systems*, Univ. Mich. Press
[18] N. Krasnogor (2002), *Studies on the Theory and Design Space of Memetic Algorithms*, Ph.D. Thesis, University of the West of England, Bristol, United Kingdom.
[19] S.A. MirHassani (2006) A computational approach to enhancing course timetabling with integer programming, *Applied Mathematics and Computation* **175**(1), 814 – 822.
[20] P. Moscato and M.G. Norman (1992), A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message– Passing Systems, *Parallel Computing and Transputer Applications*, 177 – 186.
[21] E. Ozcan (2005) Memetic Algorithms for Nurse Rostering, P. Yolum (Eds.): *Lecture Notes in Computer Science 3733*, Springer-Verlag, The 20th ISCIS, 482 – 492.
[22] E. Ozcan (2006), An Empirical Investigation on Memes, Self-generation and Nurse Rostering, *Proc. of the 6th Int. Conf. on the PATAT*, 246 – 263.
[23] E. Ozcan, E. Ersoy (2005), Final Exam Scheduler-FES, *Proc. of 2005 IEEE Congress on Evolutionary Computation* **2**, 1356 – 1363.
[24] E. Ozcan and E. Onbasioglu (2007), Memetic Algorithms for Parallel Code Optimization, *Int. J. on Parallel Processing* **35**(1), 33 – 61.
[25] B. Paechter, R.C. Rankin, A. Cumming and T.C. Fogarty (1998), Timetabling the Classes of an Entire University with an Evolutionary Algorithm, *Proc. of Parallel Problem Solving from Nature (PPSN V)*, 865 – 874.
[26] P. Ross, D. Corne and H.-L. Fang (1994), Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation, Proc. of PPSN III, 556 – 565.
[27] P. Ross, D. Corne and H.-L.Fang (1994), Fast Practical Evolutionary Timetabling, *Proc. of AISB Workshop on Evolutionary Computation*, 250 – 263.
[28] A. Schaerf (1996), Tabu Search Techniques for Large High– School Timetabling Problems, *Proc. of the Fourteenth National Conference on AI*, 363 – 368.
[29] A. Viana, Pinho de Sousa J., Matos M.A. (2003). GRASP with constraint neighbourhoods - an application to the unit commitment problem. *Proceedings of the 5th MIC.*
[30] R.F. Weare (1995), *Automated Examination Timetabling*, Ph.D. dissertation, University of Nottingham, Department of Computer Science.

Scheduling in a Multi-Processor Environment with Deteriorating Job Processing Times and Decreasing Values: the Case of Forest Fires

Costas P. Pappis, Nikos P. Rachaniotis
University of Piraeus, Department of Industrial Management and Technology, 80 Karaoli & Dimitriou str., 185 34, Piraeus, Greece, {nraxan, pappis}@unipi.gr

In forest fire fighting, time and effort required to control a fire increase if fire containment effort is delayed. The problem of scheduling multiple resources employed as parallel identical or non-identical processors in order to contain N≥2 fires may be tackled using the concept of deteriorating jobs. In this paper, the above problem is stated and a model is formulated, the criterion being to maximize the total remaining value of the burnt areas and a heuristic algorithm to tackle this NP-hard problem is proposed.

Keywords: Forest fires, scheduling, parallel processors, deteriorating jobs, multiprocessor jobs

1. Introduction

Job scheduling research has been recently extended to situations with multiple parallel identical machines (processors) and deteriorating (increasing) processing times, depending on the jobs' processing starting times. This situation appears in many real applications, e.g. scheduling industrial processors whose productivity reduces over time, scheduling industrial maintenance, repair and cleaning assignments, scheduling steel rolling mills and metallurgical processes, repayment of multiple loans, etc.

In (Chen, 1996) and (Chen, 1997) it has been shown that the parallel identical machines problem with simple linear deterioration processing times, under the criterion of minimizing the sum of completion times, is NP-hard in the ordinary sense. In (Mosheiov, 1995) heuristics were proposed for the parallel identical machines problem with linearly or stepwise deteriorating functions. In (Mosheiov, 1998) a problem with parallel identical machines and linearly deteriorating processing times was studied under the makespan minimization criterion. This problem was shown to be NP-hard and a heuristic was proposed. In (Hsieh and Bricker, 1997) three heuristics were proposed for the parallel identical machines problem with linearly decreasing processing times under the makespan minimization criterion.

Another type of problems in the area of machine scheduling is when the jobs' value is a deteriorating (decreasing) function of time, with constant processing times. The value can decrease due to several reasons like customer dissatisfaction due to delays, seasonal nature of the product, technological devaluation and/or physical degradation and lack of inventory capacity ((Smith, 1956), (Lenstra et al., 1977), (Fisher and Krieger, 1984), (Buyukkoc et al., 1985), (Kawaguchi and Kyan, 1986), (Alidaee, 1993), (Voutsinas and Pappis, 2002)).

A new version of the single-machine problem is to find the optimal scheduling policy if the processing times deteriorate, i.e., increase depending on the processing starting time, and the jobs' values also deteriorate, i.e. decrease over time. The scheduling criterion is the maximization of the total jobs' remaining value at the time that processing of all jobs is completed. Different jobs' processing times and value functions can be considered and some easy to prove properties linking this criterion with the well known minimization criterion of the sum of completion times and weighted completion times have been derived (Rachaniotis, 2004).

Regarding the above-described problem, in (Teunter and Flapper, 2003) a production-rework environment is examined. It is assumed that rework processing time and costs per item increase linearly with the time span that an item is held in stock and awaits rework. Defects

occur stochastically, and the objective is to determine the production lot size in such a way that the expected system profit be maximized. In (Inderfurth et al., 2005) a deterministic lot-size model was presented, focusing on a manufacturing system within which production as well as rework activities are carried out. Each changeover from production to rework and vice versa causes a fixed set-up cost and set-up time. Due to the deterioration caused by waiting, the cost and time of rework operations increase linearly. Considering set-up and inventory holding costs as well as set-up times, optimisation algorithms were developed in order to minimize the total cost.

Finally, a relevant application of this problem, which is tackled in (Rachaniotis and Pappis, 2006), is scheduling a single fire fighting resource when there are several forest fires to be suppressed. The time required to suppress a forest fire increases if there is a delay in the beginning of the containment's effort, while the value (monetary or of some other kind) of the affected area is assumed to decrease quadratically with the duration of the containment effort. Consequently, scheduling the available fire fighting resource in the case of multiple forest fires may be treated as a job-scheduling problem with deteriorating processing times, with the objective of maximizing the total remaining value of the burnt areas after the completion of the containment operation. The times needed for the resource to travel from one site to another were considered as set-up times (as they would be in a typical manufacturing environment). A solution methodology was proposed with water as fire suppression means. A specific model for the fire rate of spread was used and an equation for the 'processing time', i.e. the time needed for fire containment, was derived, which was related to the areas' decreasing value.

In this paper, the extension of the previously described problem of scheduling $n>1$ jobs (forest fires) on $m>1$ non-identical, parallel processors (fire fighting resources) will be examined. This is one of the main problems of the forest fire fighting operational centers in Greece and all over the world and the authors' research team has established a close co-operation with the Greek Fire Corps in the past few years. The above problem with multiprocessor jobs is stated and a model is formulated, the criterion being to maximize the total remaining value of the burnt areas and a heuristic algorithm to tackle this NP-hard problem is proposed. Multiprocessor jobs require more than one processor at the same moment of time. This is not the classical scheduling theory's usual assumption that a job is executed on one processor at a time (Drozdowski, 1996b).

2. Statement of the problem

In forest fire fighting practice, a forest region is normally assigned to several fire-fighting resources ('processors' in the sequel), which can be identical or not, e.g. water-carrying fire engines, crews with hoses connected to waterspouts, and water or retardant dropping aircraft and helicopters. A variation of Maximal Covering Location Model (MCLM), where information regarding the risk of fire ignition is embedded, may be utilized to find the proper location of the available processors in such a way as to cover optimally the considered forest region (Dimopoulou and Giannikos, 2004).

In this model, the forest region is subdivided into non-uniform, space exhausting and non-overlapping (demand) areas that must be covered. An initial value is assigned to each area, which decreases over time when a fire has ignited. This value can be monetary or ecological or of some other kinds. The value in each area decreases at a different rate, depending on the fuel type, the ecological value, whether it is inhabited or not, etc, requiring a different level of priority in a fires' suppression sequence (Rachaniotis and Pappis, 2006).

If the fire's fuel bed surface has a uniform slope and the fuel and meteorological conditions are homogeneous, then empirical evidence shows that a forest fire will expand with variable shapes, which are often approximated by ellipses with various length-to-breadth ratios. This will be the shape of the (expanding over time) fire's perimeter used in this paper. The major axis of the perimeter lies along with the wind direction. Higher wind velocities give greater Rates of Spread (ROS) and higher length-to-breadth ratios of the perimeter shape. If the

surface is sloped then the major axis of the perimeter moves upslope from that of the wind direction and lies along to what is known as "the effective wind direction" (Richards, 1999).
The primary goal of a fire containment effort is the earliest possible containment of the fire, which generally means the desperation of compatible fuel and oxygen at the fire's perimeter. This is most commonly achieved by a rapid encirclement of the fire with a fire strip, whose width depends on the intensity of the fire. In this paper this is assumed to be constant.
If more than one fire ignite either simultaneously or consecutively, it is quite possible that not all of them can be serviced as soon as they are reported. Then the decision maker must make a quick preliminary assessment of initial attack requirements and schedule each fire to a position in a priority initial attack queue. Most often, many suppression resource units are allocated to many fires when they are reported. This initial attack system may be modelled using simulation techniques as a multiserver queuing system with time dependent servicing times (Martell et al., 1984). In (Fried and Fried, 1996), a containment algorithm facilitates fire simulation with different rates of spread, tactics and resources arrivals. The outputs are the final fires' size, the containment time and the number of the resources to be utilized, provided that the fire does not escape initial attack.
The processing time, i.e. the time needed for a fire's suppression, increases (deteriorates) at a rate depending on the time elapsed from the ignition of the fire, the kind of fuel in the area that the fire burns (different fires' ROS) and the processors' fire containment efficiency. Consequently, since time and effort required to control a forest fire increases with time, scheduling the available fire fighting resources in the case of multiple forest fires may be treated as a job-scheduling problem with deteriorating job processing times and decreasing job values. The objective is the minimization of the total damage caused by the fires to the burnt areas or, equivalently, the maximization of the total remaining values of the areas.
This situation involves some special characteristics, including the following:

a. The parallel non-identical processors are unrelated and their containment rate is job-dependent (fire-dependent).
b. Fires can be considered as multiprocessor jobs, i.e. more than one processor is required at the same time for a specific fire.
c. Pre-emption, i.e. the situation where the processor is called to suppress a fire while it is busy suppressing another one, is allowed.
d. Jobs have variable profiles, i.e. the number of processors employed for specific fire suppression can change during the containment effort.

2.1 Notation

Let:
F: be a set of N forest fires $\{F_1,\ldots,F_N\}$ with $L\leq N$ different values of ignition times ("release times"), $0=r_1<r_2<\ldots<r_L$.
$V_i(t)$: be the remaining value of the area that F_i burns at time t.
V_{i0}: be the initial value of the area that F_i burns.
R: be the set of fire fighting kinds of resources $\{R_1,\ldots,R_K\}$, i.e. it is assumed that there are K different subsets (kinds) of non-identical resources with M_j identical processors of each kind, j=1, 2,…,K, and $\sum_{j=1}^{K} M_j = M$, i.e. M is the total number of the parallel unrelated processors (identical or non identical).
$t_0>0$: be the time instance of the beginning of the forest fire suppression effort.
$\rho_{ij}(t)$: be the containment rate of F_i by resource of kind j at time t.
$\underline{s}_i(t) = (s_{i1}(t),...,s_{iK}(t))$: be the vector of the numbers of resources employed for the suppression of F_i at time t, i=1,2,…,G≤N.
$P(\underline{s}_i(t),t)$: be the processing time, i.e. the time needed to contain F_i at time t.
$Area_i(t)$: be the area of the elliptical fireline strip that has to be created in order to contain F_i at time t.

α,β,γ,ε: be constant parameters depending on the kind of the area that F_i burns (the fires' ROS differs in different demand areas) and the meteorological conditions.
$t_{max,i}$: The "containment escape time limit" of F_i, i=1,2,…,N. F_i has escaped initial attack if its containment time exceeds $t_{max,i}$. It can be calculated at the ignition time of F_i.
$\underline{\delta}_i = (\delta_{i1},...,\delta_{iK}): \quad P(\underline{\delta}_i, r_i) = t_{\max,i}, with\ \delta_{ij} \le M_j$
be the vector of "minimal requirements" for the suppression of F_i, i.e. the necessary resources in order to contain F_i in $t_{max,i}$ and not to consider it as an escaped initial attack fire.
$h_i(t)$: be the shortest time required to suppress F_i at any time t. In this paper it will be called the *height* of F_i at time t.
$C(\underline{s}_i(t)): h_i(C(\underline{s}_i(t)) = 0$ be the completion time of F_i's suppression

2.2 Assumptions

a. The available processors fully cover the region examined and are located inside it.
b. The processors can always reach any fire.
c. The fuel and meteorological conditions remain constant (homogeneous conditions) at each demand area.
d. The burn time of an area depends on its fuel and is not affected by the fuel of adjacent areas.
e. The value of the burning area that F_i burns deteriorates quadratically over time with a constant rate w_i, i.e.
$$V_i(t)=V_{i0}-w_i\, C^2(\underline{s}_i(t))$$
f. In every fire containment effort there is a number of time intervals where the assignment of fires to resources is constant.
g. Different resource types can substitute one another.

3. The model

Fireline is rarely built at a constant rate for the duration of an initial attack effort. The productivity decreases as resources' traveling times ("set-up times" in a typical job-scheduling problem in a manufacturing environment) are included, crews' fatigue appears and water and retardant supplies are depleted. This is called 'drop-off' phenomenon (Fried and Fried, 1996). The result is a stepwise containment rate function:

$$\rho_{ij}(t) = \begin{cases} a_{ij}, t \le t_{ij} \\ b_{ij}, t > t_{ij} \end{cases}, b_{ij} < a_{ij}$$

where t_{ij} change as the containment effort "clock time" increases.
The processing time equation is ($Area_i(t)$ is calculated in Rachaniotis and Pappis, 2006):

$$P(\underline{s}_i(t), t) = \frac{Area_i(t)}{\sum_{j=1}^{K} s_{ij}(t)\rho_{ij}(t)} = \frac{\alpha\beta^2 t^{2\gamma} + 2\alpha\beta\varepsilon t^{\gamma+1} + \alpha\varepsilon t^2}{\sum_{j=1}^{K} s_{ij}(t)\rho_{ij}(t)}$$

The height of F_i at time t (defined as the area of the elliptical fire fighting strip that is not contained at time t divided by the maximum containment rate) is calculated using the equation

$$h_i(t) = \frac{Area_i(t) - \int_{t_0}^{t} P(\underline{s}_i(\tau), \tau) \sum_{j=1}^{K} s_{ij}(\tau)\rho_{ij}(\tau) d\tau}{\sum_{j=1}^{K} M_j \rho_{ij}(t)}$$

The initial height of F_i (for $t=t_0$) is

$$h_i(t_0) = \frac{Area_i(t_0)}{\sum_{j=1}^{K} M_j \rho_{ij}(t_0)}$$

The objective is to minimize the total damage caused in the burned areas,

$$\min \sum_{i=1}^{N} w_i C^2(\underline{s}_i(t))$$

or equivalently to maximize their total remaining value:

$$\max \sum_{i=1}^{N} V_i(t)$$

4. Heuristic Algorithm

4.1 Algorithm's description

The heuristic presented here for tackling the aforementioned problem uses some ideas of Drozdowski's algorithm. (Drozdowski, 1996a) examined a problem of scheduling parallel applications in a multiprogrammed multiprocessor system. The preemptive case was addressed with a variable number of processors used by a task over time. A low-order polynomial time algorithm was proposed for minimizing the makespan, which schedules tasks according to their "level" which is the time required to finish them.

The intervals between the fires' ignition times are considered consecutively. Subintervals are created where the ratios $w_i/h_i(t)$ are not changing. The rational for using these ratios stems from the well known Smith's rule for solving optimally the problem $N/1/min\Sigma a_iC_i$ with constant processing times P_i, which is to schedule jobs in non-decreasing order of P_i/a_i (Smith, 1956). This heuristic is also used in the problem $N/1/min\Sigma a_iC_i^2$ with constant processing times, which is a much more simple but quite similar to the problem examined in this paper ((Della Croce et al., 1995), (Mondal and Sen, 2000)).

Fire suppressions are assigned to processors, and fires with higher ratios are given preference. When there are more processors available than those, which are simultaneously required by the fires already ignited, a maximal possible number of processors are assigned. Otherwise, processors are shared by fires with equal ratios so that their heights decrease at the same rate. The time length of the current assignment is calculated. This assignment of processors changes in three cases: a) The height for some fire with initially higher ratio becomes equal to the height of some fire with initially lower ratio and the processing times must be recalculated in order to obtain the same rate of height decrease for equal ratios fires, b) The fire suppression with the shortest completion time in the release times' interval finishes or c) The end of the interval is encountered and fire suppression tasks in the next interval must be considered. Finally, McNaughton's wrap-around rule (McNaughton, 1959) is used to schedule pieces of fire suppression tasks. This rule uses a new processor for containing a fire only if the previous processor is fully employed for the examined time interval.

Note that no information about fire ignitions in the next release times' intervals is necessary to schedule the suppression of fires with release times less than r_{k+1}. Hence, the above algorithm is a real-time (dynamic) algorithm and it is synchronous (can be run on-line), meaning that it builds sub-optimal schedules using only the information about fires that have already ignited. The algorithm's validity proof is similar to the one in (Drozdowski, 1996).

A schematic overview of the algorithm, which is analytically presented in the next sub-section, is the following:

Step1: Schedule fires in non-ascending order of ratios $w_i/h_i(t)$.
Step2: Assign resources to fires already ignited.
Step3: Check for any needs in altering resources' assignments.
Step4: Use McNaughton's rule for scheduling pieces of fire suppression tasks.
Step5: Calculate new fires' heights.

4.2 The algorithm

$t=t_0$
for k=1 to L do
  Set_k=set of fires with ignition time r_k

Schedule fires in Set_k in non-ascending order of ratios $w_i/h_i(t)$. Then the fires are denoted $F_{[1]},F_{[2]},\ldots$
while ($r_{k+1}>t$) and ($\exists\ F_i \in Set_k$: $h_i(t)>0$) do
(*procedure of assigning resources*)
$\underline{s}(t) = \underline{0}$
$avail_j=M_j$ ($\Sigma M_j=M$) (*available processors of kind j *)
while ($avail_j>0$ for some kind j) and $|Set_k|>0$ do
if $\sum_{F_{[i]} \in Set_k} \delta_{[i]j} > avail_j$ then
for i=1 to $|Set_k|$ do
for j=1 to K do
$s_{[i]j}(t)=\delta_{[i]j}\cdot$($avail_j$ with highest containment rates)/$\sum_{F_{[i]} \in Set_k} \delta_{[i]j}$
$avail_j=0$
else
for i=1 to $|Set_k|$ do
for j=1 to K do
$s_{[i]j}(t)=\delta_{[i]j}$ with highest containment rates

$$\text{avail}_j = \text{avail}_j - \sum_{i=1}^{|Set_k|} \sum_{j=1}^{K} \delta_{[i]j}$$

(* end of procedure*)
if $\exists\ F_{[i]}, F_{[q]} \in Set_k$: $w_{[i]}/h_{[i]}(t) > w_{[q]}/h_{[q]}(t)$ then
calculate $t_1=\min\{t: w_{[i]}/h_{[i]}(t) = w_{[q]}/h_{[q]}(t)$ for every pair of jobs $F_{[i]}, F_{[q]} \in Set_k$
(*the shortest time required for 2 jobs different ratios to become equal*)
else t_1=A, A arbitrary large number
$t_2 = \min_{F_{[i]} \in Set_k} \{C(\underline{s}_{[i]}(t))\}$
(*This is fire's $F_{[i]}$'s shortest suppression completion time, i.e. $h_{[i]}(C(\underline{s}_{[i]}(t_2), t_2)) = 0$ *)
$t_{dec}=\min\{t_1,t_2, r_{k+1}-t\}$
(* Schedule $t_{dec} \sum_{j=1}^{K} s_{[i]j}(t)\rho_{[i]j}(t)$ "piece" of fire's $F_{[i]}$ suppression in the interval [t, $t+t_{dec}$]
according to McNaughton's wrap-around rule (change the assignment of resources to fires) *)
for i=1 to $|Set_k|$ do
if there is no resource "partially" empty and a non-busy resource exists then
assign fire $F_{[i]}$ to a non-busy resource
else
calculate for all processors the x unscheduled units of time until $t+t_{dec}$
if $P(\underline{s}_i(t+t_{dec}), t+t_{dec}) \le x$, then assign fire $F_{[i]}$ to the available processor with the highest containment rate
else
this resource is busy on [t, $t+t_{dec}$-x]
assign fire $F_{[i]}$ to the processor with the next highest containment rate for [t, $P(\underline{s}_i(t+t_{dec}), t+t_{dec})$] (if possible) or [t, $t+t_{dec}$-x] and then assign fire $F_{[i]}$ to the processor with the highest containment rate for [$t+t_{dec}$-x, $t+t_{dec}$] (if not possible)
$t=t+t_{dec}$
for i=1 to $|Set_k|$
Calculate new $h_{[i]}(t)$
if $\exists\ F_{[i]} \in Set_k$: $h_{[i]}(t) > 0$ then
$Set_{k+1}=Set_{k+1} \cup \{F_{[i]}\}$

6. Conclusion

Scheduling the available fire fighting resources in the case of multiple forest fires may be treated as a job-scheduling problem with deteriorating job processing times and decreasing job values. The problem was stated as a multiprocessor jobs one. A model has been formulated with optimization criterion to maximize the total jobs' remaining value. A heuristic algorithm was presented in order to tackle this NP-hard problem.
The next research steps may include: a) implementation of the heuristic algorithm and testing its efficiency compared to real forest fires' data provided by Greek Fire Corps and b) use of $\Sigma w_i T_i$ or N_T as optimality criteria, where T_i is the tardiness (if any) in suppressing fire F_i and N_T the number of tardy fire containment efforts. Finally the model could be applied to other real life situations such as epidemics control.

References

[1] B. Alidaee (1993), Numerical Methods for Single machine Scheduling with Non-linear Cost Functions to Minimize Total Cost, *Journal of the Operational Research Society* **44**, 125 – 132.
[2] C. Buyukkoc, P.Varaiya, J. Walrand (1985), The cμ Rule Revisited, *Advanced Applied Probability* **17**, 237 – 238.
[3] Z. L. Chen (1996), Parallel machine scheduling with time dependent processing times, *Discrete Applied Mathematics* **70**, 81 – 93.
[4] Z-L. Chen (1997), Erratum to "parallel machine scheduling with time dependent processing times", *Discrete Applied Mathematics* **75**, 103.
[5] F. Della Croce, W. Szwarc, R. Tadei, P. Baracco, R. Di Tullio (1995), Minimizing the weighted sum of quadratic completion times on a single machine, *Naval Research Logistics* **42**, 1263 – 1270.
[6] M. Dimopoulou, I. Giannikos (2004), Towards an integrated framework for forest fire control, *European Journal of Operational Research* **152**(2), 476 – 486.
[7] M. Drozdowski (1996a), Real-time scheduling of linear speedup parallel machines, *Information processing Letters* **57**, 35 – 40.
[8] M. Drozdowski (1996b), Scheduling multiprocessor tasks- An overview, *European Journal of Operations Research* **94**, 215 – 230.
[9] M.L. Fisher and A.M. Krieger (1984), Analysis of a linearization Heuristic for Single-Machine Scheduling to Maximize Profit, *Mathematical Programming* **28**, 218 – 225.
[10] J. S. Fried, B. D. Fried (1996), Simulating wildfire containment with realistic tactics, *Forest Science* **42**(3), 267 – 281.
[11] Y. Hsieh and D.L. Bricker (1997), Scheduling linearly deteriorating jobs on multiple machines, *Computers and Industrial Engineering* **32**(4), 727 – 734.
[12] K. Inderfurth, G. Lindner, N.P. Rachaniotis (2005), Lotsizing in a Production System with Rework and Product Deterioration, *International Journal of Production Research* **43**(7), 1355 – 1374.
[13] T. Kawaguchi and S. Kyan (1986), Worst Case Bound of an LRF Schedule for the Mean Weighted Flow-Time Problem, *SIAM Journal of Computing* **15**, 1119 – 1129.
[14] J.K. Lenstra, R. Kan and P. Brucker (1977), Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics* **1**, 343 – 362.
[15] D. L. Martell, R. J. Drysdale, G. E. Doan and D. Boychuk (1984), An evaluation of forest fire initial attack resources, *Interfaces* **14**(5), 20 – 32.
[16] R. McNaughton (1959), Scheduling with deadlines and loss functions, *Management Science* **6**, 1 – 12.
[17] S.A. Mondal, A.K. Sen (2000), An improved precedence rule for single machine scheduling problems with quadratic penalty, *European Journal of Operational Research* **125**, 425 – 428.
[18] G. Mosheiov (1995), Scheduling jobs with step-deterioration: Minimizing makespan on a single and multi-machine, *Computing and Industrial Engineering* **28**, 869 – 879.

[19] G. Mosheiov (1998), Multi-machine scheduling with linear deterioration, *INFOR* **36**, 205 – 214.
[20] N.P. Rachaniotis (2004), Advances on job scheduling with time dependent parameters: theory and industrial applications, *Electronic proceedings of the 9th ELA doctorate workshop, 16th-18th June 2004, Monchy St. Eloi, France.*
[21] N.P. Rachaniotis, C.P. Pappis(2006), Scheduling Fire Fighting Tasks Using the Concept of "Deteriorating Jobs", *Canadian Journal of Forest Research* **36**, 652 – 658.
[22] G.D. Richards (1999), The Mathematical Modelling and Computer Simulation of Wildland Fire Perimeter Growth Over a 3-Dimensional Surface, *International Journal of Wildland Fire* **9**(3), 213 – 221.
[23] W.E. Smith (1956), Various Optimizers for Single-Stage Production, *Naval Research Logistics Quarterly* **3**, 59 – 66.
[24] R.H. Teunter and S.D.P. Flapper (2003), Lot-sizing for a single-stage single-product production system with rework of perishable production defectives, *OR Spectrum* **25**, 85 – 96.
[25] T.G. Voutsinas, C.P. Pappis (2002), Scheduling jobs with values exponentially deteriorating over time, *International Journal of Production Economics* **79** (3), 163 – 169.

A Window Time Negotiation Approach at the Scheduling Level inside Supply Chains

Marie-Claude Portmann, Zerouk Mouloua
LORIA-INPL, Ecole des Mines de Nancy, Parc de Saurupt, CS 14234, 54042 Nancy Cedex, France,
{portmann, mouloua}@loria.fr

We consider a supply chain, which consists in a network of enterprises with independent decision centers. The finished products (or sub products) of the assembly enterprise are produced using components and/or sub products supplied by other enterprises or by external suppliers. We assume we are at the scheduling level and each enterprise builds its own schedules associated with its own production centers. As an operation can be performed only when the production center has received the necessary components and/or sub products, the schedules are dependent. This induces negotiations between decision centers of enterprises, which can be expressed in term of penalty functions associated with soft and hard release dates and due dates. At each negotiation point, hard release dates and due dates are considered as imperative constraints, while soft release dates and due dates define soft intervals and induce earliness and tardiness penalties. Both soft and hard constraints can be modified during the negotiation process. A global solution is searched by an iterative decomposition approach including alternatively bilateral negotiations of the soft and hard constraints between the production decision centers and just in time scheduling, minimizing the local total sum of penalties, built by approximation approaches. We assume that production centers are flow shop (linear production and/or assembly line). To solve each local just-in-time scheduling problem, we propose an approximation approach based on meta-heuristics, which explores the set of feasible and infeasible solutions, in which a solution is described by the job order on each machine (permutation of integers) and is evaluated using a "pert cost" algorithm. The infeasible solutions are evaluated by a lower bound of the number of non verified imperative constraints and the feasible solutions are evaluated by the minimal sum of penalties corresponding to the considered order. A semi-decentralized control is suggested to assume the negotiation convergence.

Keywords: scheduling, supply chain, just in time, negotiations, decomposition approach, "pert cost" algorithm, meta-heuristics

1. Introduction

We consider a supply chain, which consists in a network of enterprises with independent decision centers. In consequence, we assume each decision center can prepare its own schedules and negotiate release dates and due dates with their suppliers (upstream level of the supply chain) and their customers (downstream level of the supply chain). The decision system could have been totally decentralized, but in this case, there is no guarantee of convergence of the global negotiation process. In consequence, we choose a semi-decentralized decision system in which a common memory is shared and global rules are agreed at the network level in order to help the negotiations to converge (eventually to infeasible solutions of the initial global just in time scheduling problem, by authorization of unlimited increase of the finished product tardiness in the worst cases).
We consider the following production system. The finished products (or sub products) of the assembly enterprise are produced using components and/or sub products supplied by other enterprises or by external suppliers. We assume we are at the scheduling level and each enterprise builds its own schedules associated with its own production centers. We assume the transport between production centers is taken into account only by minimal time lags. As an operation on a product can be performed only when an enterprise has received the needed components and sub products, the schedules are in fact dependent. This induces negotiations between enterprises, which can be expressed in term of soft and hard release dates and due dates. Hard release dates and due dates can be changed in order to increase or decrease the set of feasible solutions of each local

partner. Soft release dates and due dates and their associated earliness and tardiness penalties can be real or only virtual delays and penalties in order to lead the whole network towards a win-win global solution. A global solution is searched by an iterative decomposition approach including alternatively bilateral negotiations of the soft and hard delays between the production centers and just in time scheduling minimizing the total sum of penalties inside each decision or production center. A semi-decentralized global negotiation approach will be suggested in section 2.
We assume that production centers are either flow shop (assembly line) or even job shop (this complicates only the just in time scheduling algorithms). To solve each local just-in-time scheduling problem associated with a given set of hard and soft release dates and due dates and a given set of penalty data, we propose an approximation approach based on meta-heuristics, which explores the set of feasible and infeasible solutions, in which a solution is described by the job order on each machine (permutation of integers) and is evaluated, using a polynomial "pert cost" algorithm. Section 3 presents the just in time scheduling problem and how the timing problem can be solved by a "pert cost" algorithm. Some experiments concerning the just in time scheduling problem are presented at the end of section 3.

2. Global negotiation model

2.1. Global problem description

In the framework of decentralized or semi-decentralized decision system inside supply chain, several models have been proposed at the medium term level (Dudek and Stadtler, 2005), (Hurtubise et al, 2004). This proposition is in continuation of our own research works at the medium term planning (Ouzizi et al, 2006). At the medium term level, volumes associated with family of finished products, sub products and components are forecasted and production and transportation phases are planned in order to take into account the limited capacities of the resources. Negotiations can be organized between the independent partners in order to take into account their limited capacities and aggregated linear models can be used in order to compute each local planning. At the medium term level, hard and soft cumulated curves of component arrival or product delivery are negotiated between the supply chain partners and complementary resources can be bought. At the scheduling level, the forecasted medium term volumes on product family have been converted into firm orders of very precisely definite products. Each order is associated with one or several jobs (if lot sizing or splitting is used). Hence a job corresponds either to only one product or to an inseparable series of products, for which the detail schedule parameters are known: production centers to be visited, main sub products to be waited for (release date on the first machine of a given shop or further on another assembly machine), one or more production process inside each center (generalized job shop or hybrid flow shop). We assume here that external due dates have been promised to the final customers, which do not belong to the supply chain, and external release dates have been agreed with the external suppliers, most of them being imperative, due for example to transportation delays. We had to find the global schedule of the considered supply chain, while minimizing the total production cost and the total external penalties: win-win strategy inside the supply chain.

2.2. Global negotiation process

The ideas used at the medium term level, i.e. hard and soft cumulated curves of component arrival or product delivery with earliness and tardiness penalties negotiated between the supply chain partners, could be extended to the scheduling level replacing hard and soft cumulated curves by hard and soft operation windows. The main idea of the global negotiation process is also an extension of the process proposed in (Portmann, 1988) and (Chu et al, 1992). In these papers, the criterion considered was the sum of weighted tardiness (a regular criterion) and only active schedules associated with hard release dates were considered; in consequence, only hard release dates, but hard and soft due dates were used in this iterative splitting approximation approach, whose aim was to

build a global schedule for a given production center, obtained by scheduling optimally cells and modifying hard release dates and hard and soft local due dates. This splitting approach gave very efficient solution methods.
The extended similar process is the following one illustrated by a small example on one product.

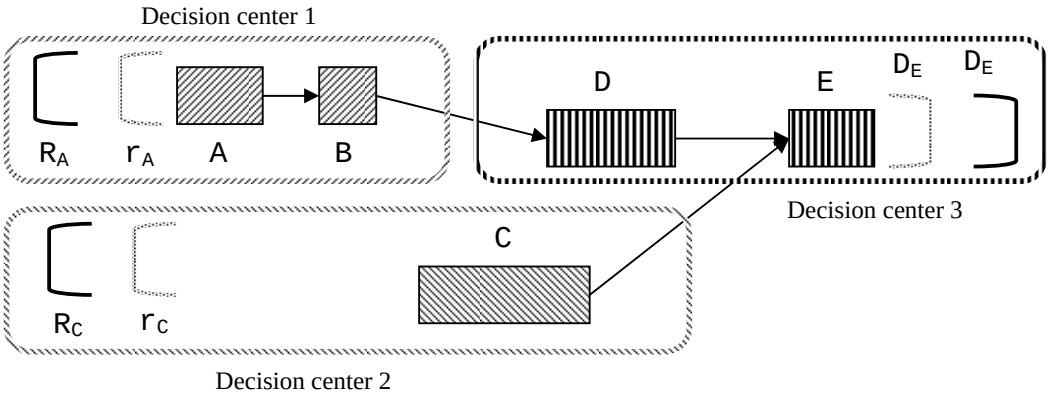


Figure 1: external constraints

In figure 1, soft and hard due dates associated with a given finished product order due to an external customer are denoted by d_E and D_E. The assembly of this finished product order needs two types of sub products used at two successive operations of the assembly line and sub products, made by two different supply chain partners, need themselves components from external suppliers, whose soft and hard release dates are respectively r_A, R_A, r_C and R_C.
If there exists only one decision center for the whole supply chain, then we have got only one just in time scheduling problem to be solved, in which production costs must be minimized while verifying the external hard release dates and due dates (if possible) and minimizing the total sum of earliness and tardiness external penalties.
While we consider a semi-decentralized control of independent decision centers, each decision center computes its own just in time scheduling and negotiates hard and soft release dates with the other partners of the supply chain with the aim of finding a global solution as good as possible, which will be a trade-off between the interests of the partners.
To organize the negotiations, we need initial soft and hard window constraints between the whole set of partners. They will be called "virtual"; because they are not imposed by the global just in time scheduling problem and they will change during the global negotiation process.

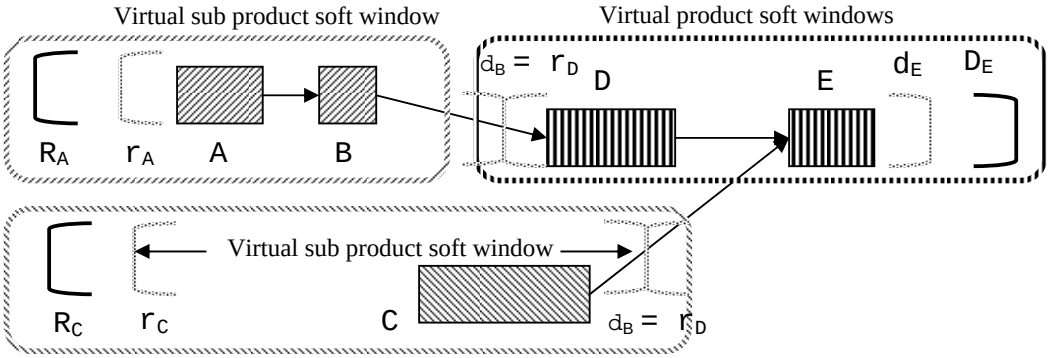


Figure 2: initial soft window constraints

First, similarly to (Portmann, 1988), we use the soft release dates and due dates of each sub product, of the final product and their processing times and we call a sharing function (for example, soft window lengths are proportional to the corresponding sum of processing times) in order to define "virtual" windows for each manufacturing part (see figure 2). It can be remarked that re-

lease and due dates are associated to any operation, whose predecessors and/or successors are in another decision centers (minimal time lags can be added in the model for transportation duration). Hard initial windows are also obtained by a given rule. For example, they can be very large at the beginning of the global process (each operation is left shifted on its own hard release date or on its previous left shifted operations for the hard release dates and each operation is right shifted on its own hard due date or on its following right shifted operations for the hard due dates) or the difference between the hard and soft window lengths of each sub product and/or product is computed by a function similar to the one use for the soft constraints (proportionality rule). Virtual penalties are attributed equally without any further information or individually increased when, for example, it is known that some machine of the production system is overloaded and upstream and downstream operations must respect as tight as possible the virtual given windows.

After this hard and soft windows attribution, the schedules of the various production centers become independent and an exact or approximation scheduling tool can be used in order to minimize the earliness and tardiness penalty sum while respecting the hard windows. Each schedule can be computed locally by the decision centers in real life applications. This initial schedule computation finishes the first step of the global process.

An analysis is made at the end of each step of this splitting approach.
Case 1: If the external soft release dates and due dates are respected and any precedence constraint between two different production centers is respected, i.e. when an operation is late relatively to hard or soft “virtual” due dates, then we have the chance that the following operations are at least as late as it and when an operation is early relatively to its hard or soft “virtual” release date then the previous operations are sufficiently early for the global schedule to be feasible. It is the perfect case and the negotiation process can stop without any external negotiations.
Case 2: If external hard release dates and due dates are respected, but not any soft external windows, and nevertheless the schedules of the various production centers are compatible, then two ways can be followed before the next step, either to negotiate new external soft release dates and/or due dates or to increase the penalty associated with the external earliness and tardiness hoping to get a better set of feasible local schedules.
Case 3: If external hard windows are respected, but the set of schedules induce incompatibility between production centers, then the internal hard and soft windows must be modified in order to converge to a set of feasible schedules. In a semi-decentralized decision system approach, the order in which the windows are negotiated can be organized, going from supplier to customer (direct succession of schedules and windows modification) or from customer to supplier (retrograde succession of schedules and windows modification, knowing that the minimal hard release date must remain always positive). This organization is a way to assume the convergence of the global process (assuming that case 4 can enlarge the external hard windows to increase the set of feasible solutions).
Case 4: If external hard windows are not respected, it is a very delicate situation. The explanation could be that the whole global problem has no feasible solution, but it is impossible to verify if the problem is too big to use an exact algorithm to solve the centralized global just in time scheduling problem. The only way to continue is to relax slightly the hard final due dates and to begin a new negotiation process between the partners.
The convergence of the process can be assumed by the organization of the authorized negotiations (no loop in the negotiation process) and by the last assumption that hard final due dates can be relaxed. The efficiency of this global negotiation approach can only be verified by experimentation using simulation for modeling the human behavior inside the decision process.

3. Just in time scheduling problems

To complete this window negotiation process, we need a tool for solving the scheduling problem with hard and soft window inside each production center. We simplify here the scheduling prob-

lem assuming there are no production costs (such as inventory costs or overtime hours or sub contracting …) to be considered and we have only to minimize the sum of earliness and tardiness penalties.
If the local shop we consider is a flow shop and hard and soft release dates are associated to the first job operations, while hard and soft due dates are associated to the last job operations, then our problem can be denoted by $F/\overline{r_i}, r_i, d_i, \overline{d_i}/\Sigma\alpha_{r,i}E_i^r + \Sigma\beta_{r,I}T_i^r + \Sigma\alpha_{d,i}E_i^d + \Sigma\beta_{d,I}T_i^d$ in the classical $\alpha/\beta/\gamma$ notation where F means flow shop, $\overline{r_i}, r_i, d_i, \overline{d_i}$ are respectively the hard and soft release dates and due dates of the first and last operations of each job and $\alpha_{r,i}$, $\beta_{r,i}$, $\alpha_{d,i}$, $\beta_{d,i}$ are the earliness (α) or tardiness (β) penalty associated with the release date of the first operation ($_r$) and the due date of the last operation ($_d$). When we assume that release dates and/or due dates can be associated to any operations of the jobs, we can use a more general $\alpha/\beta/\gamma$ notation: J / $tmin_j$, t^*_j, $tmax_j$ / $\Sigma\alpha_jE_j+\Sigma\beta_jT_j$, where t^*_j is the ideal position of the starting time of the operation j, $tmin_j$ is its minimal authorized value (hard constraint for earliness) and $tmax_j$ is its maximal authorized value (hard constraint for tardiness).
We propose to use another approximation decomposition approach to solve this local problem. If we assume the operation sequence is given on each machine (without circuit for the job shop problem), then we have only to compute the optimal start time of each operation (i.e. where are put the idle times, because our criterion is not regular) and we use a meta-heuristic approach to explore the set of operation sequences on the machine in order to find a feasible solution as good as possible. We now present the timing scheduling problem when the sequence is given (by a chromosome) and the main feature of the meta-heuristic afterwards.

3.1. Timing scheduling problem (i.e. chromosome evaluation)

We have to solve the $F_{seq}/\overline{r_i}, r_i, d_i, \overline{d_i}/\Sigma\alpha_{r,i}E_i^r + \Sigma\beta_{r,I}T_i^r + \Sigma\alpha_{d,i}E_i^d + \Sigma\beta_{d,I}T_i^d$ scheduling problem or the J_{seq} / $tmin_j$, t^*_j, $tmax_j$ / $\Sigma\alpha_jE_j+\Sigma\beta_jT_j$, where "seq" means the sequences on the machine are given by a chromosome. For job shop problem, some chromosomes can be unacceptable whatever could be the hard windows because of circuit in the precedence graph, while this cannot happen for flow shop or single machine problem. We assume the meta-heuristic used is able to build only chromosome with no circuit in the precedence graph (we have not this problem in our first experiments because we consider only flow shops). When considering only the hard windows ($\overline{r_i}$, $\overline{d_i}$) or [$tmin_j$, $tmax_j$], for fixed sequence on the resources, our problem is simply a "PERT" problem, which could be polynomially solved by computing the left shifted schedule ($\lambda_{k,i}$) or (λ_j) and the right shifted schedule ($\lambda'_{k,i}$) or (λ'_j) and verifying that for each operation k of each job i (denoted j) we have $\lambda_j \leq \lambda'_j$. We use the number of times this inequality is not verified to measure the infeasibility of the chromosomes relatively to the hard windows, but we keep this infeasible chromosome with very bad fitness as current solution of simulated annealing or inside the genetic algorithm population.

We now assume a chromosome is totally feasible (no circuit due to the sequences on the machine and there exists at least one solution verifying the hard windows) and we want to find each operation starting time of a feasible schedule in order to minimize the total sum of earliness and tardiness penalty. This problem can be modeled as a "pert cost" scheduling problem as follows. s is the source and t the sink of the "pert cost" extended graph CG, where each operation, assumed to be non preemptive can be represented by its beginning event. The duration of each operation is added to the value of the arc issued from this event. The operation sequence of the job and on the machine is represented by usual precedence constraints between the operations.
We denote by H the greatest hard due date (or the greatest $tmax_j$), H is the length of the considered horizon. The figure 3 gives the sub graph representing earliness or tardiness penalty for any operation. OP_j is the beginning of the operation j, t^*_j is its ideal position. If x_i is equal to 0, then the

operation j is ideally placed. If x_i is positive, then the event is early and if it is negative then the event is late. We considered that the arc issued from s is compressed and induce a cost $\alpha_j.x_j$ when x_j is positive and that the arc arriving in t is compressed and induce a cost - $\beta_j.x_j$ when x_j is negative. It is important to remark than the length of the path from s to t through BS_j, OP_j and ES_j is exactly equal to H, the length of the horizon and the values x_j will be chosen so that the maximum path in the complete precedence graph will also be equal to H, and in this case, any operation j is placed on the time axis by knowing the value of x_j. $t^*_j - x_j$ must remain greater than $tmin_j$ and smaller than $tmax_j$.

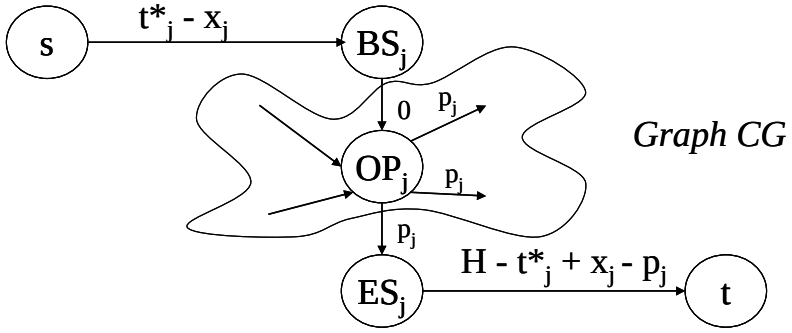


Figure 3: precedence constraints linked to earliness and tardiness penalties

When points of the graph CG are in series with the same set of successors or predecessors, we can delete the beginning of the point series and add the values of the delete arcs to the last arc kept. The data of a particular instance corresponding to a flow shop with two machines and earliness and tardiness cost relatively to the release dates for the first machine and the due dates for the second machine are given in table 1 and the simplified corresponding "pert cost" extended graph is given in the figure 4.

i	p_1	p_2	$\overline{r}$	r	α_r	β_r	α_d	β_d	d	$\overline{d}$
1	5	2	0	10	2	1	1	2	20	30
2	2	3	3	9	3	2	1	3	15	30
3	4	5	1	7	3	0	0	5	18	30
4	4	2	5	9	4	3	2	3	25	30
5	3	1	3	5	2	1	1	2	22	30

Table 1: precedence constraints linked to earliness and tardiness penalties

It is a particular "pert cost" problem in which you can only pay to decrease the duration of the arcs adjacent to the source s or to the sink t. You can solve it polynomially using a dual approach. But due to the particular shape of the graph, the primal algorithm (using alternatively "pert" algorithm and search of minimal cut on the critical sub graph), when using the maximal compression associated to each cut, is probably polynomial (we do not prove it formally until now). Any way, the minimal sum of earliness and tardiness penalty can be computed for each chromosome by using a "pert cost" algorithm and this sum is used to compute the fitness of each chromosome.

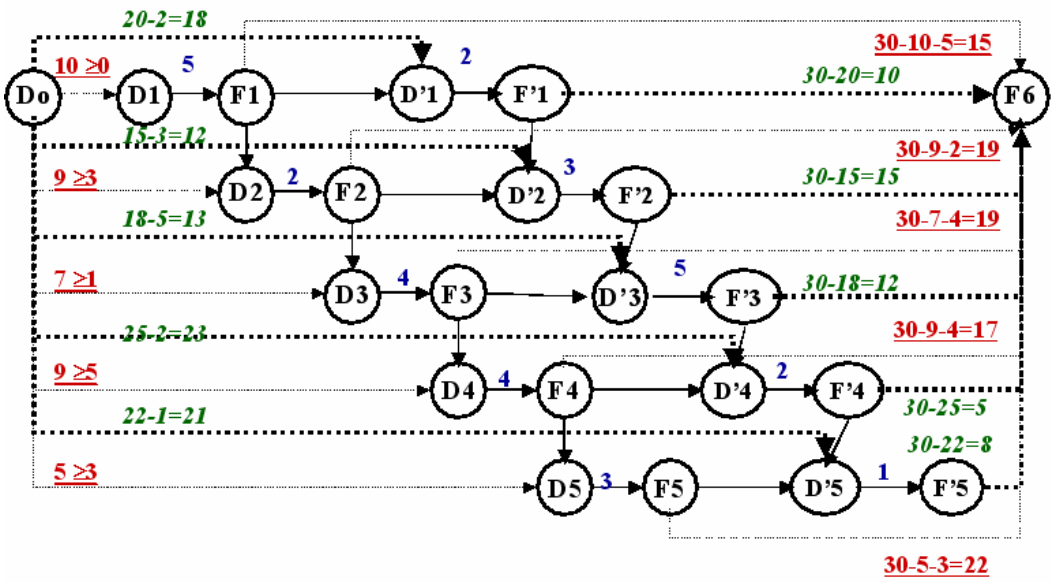


Figure 4: simplified extended "pert cost" for a flow shop problem

3.2. Meta-heuristic approaches

We already used and published papers using meta-heuristic and more specifically genetic algorithms for which we analyze specifically the performance of the permutation cross-over operators for various criteria (Djerid and Portmann, 2000), (Portmann and Vignier, 2000). Nevertheless a rapid experimentation using the simple genetic algorithm (SGA) of (Goldberg, 1989) provides us with deceiving results and our genetic algorithm needs to be improved. On another way, simulated annealing is very rapid to test and only to show meta-heuristic could be a good approach for our problem we use it also in the following results.

3.3. First experimentation for the just in time scheduling problem

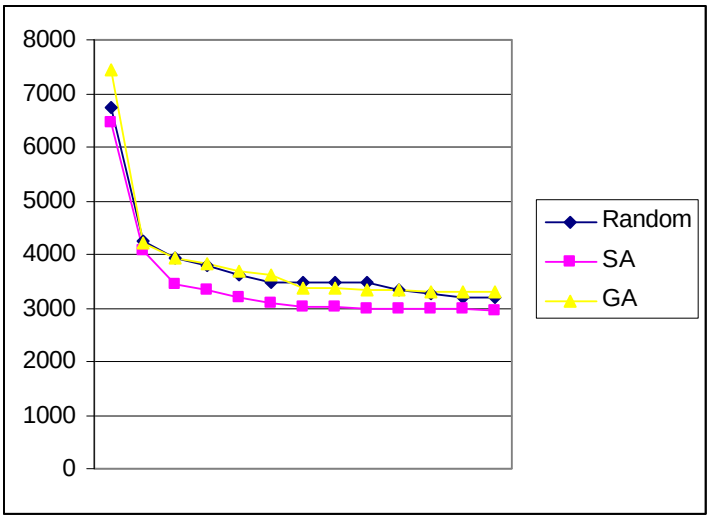


Figure 5: comparison of approximation approach

The experiments were made on generated flow shop examples with 10 jobs and 5 machines. For each instance, three algorithms were used: random generation of a sequence (Random), simulated annealing (SA) and simple algorithm genetic (GA). To see the rapidity to find solution as good as

possible, we compute the function BEST(Algo, k) which gives the value of the best found feasible solution after having evaluated k sequences. To eliminate the fact that hazard could be more or less favorable, we execute each algorithm 5 times on the same instance and we keep the mean value, which provides us with MEAN-BEST(Algo,k) and we present here in the figure 5 the average on four instances. Only 300 feasible solutions analyzed by "pert cost", where generated for each execution of the algorithm. Bad results obtained by the genetic algorithm can be due to the fact that the initial population is randomly built and it is possible, with only 300 sequences studied among 10!, SA or GA have not yet converged. We will continue our experimentation with longer computational time and with improvement of our very simple current meta-heuristic scheme.

4. Conclusion

We propose here a new decomposition approach to organize the negotiation between various independent partners of a supply chain at the scheduling level using hard and soft windows as entity of negotiation. This decomposition can also be used as an approximation approach for building a just in time solution in presence of only one decision center and several production centers, as an extension of the method proposed in (Chu et al, 1992). The interest of this last proposition is that it can be compared with other methods using experimentations, while the model with several decision centers and human decision models must be validated using simulation. Another important problem will be to convince the industrial decision makers and managers of the interest of the proposed models.

References

[1] C. Chu, M.C. Portmann, J.M. Proth (1992), A splitting up approach to simplify job-shop scheduling problems. *International Journal of Production Research*, **30**(4), 859 – 870.

[2] L. Djerid, M.C. Portmann (2000), How to Keep Good Schemata Using Cross-over Operators for Permutation Problems. *International Transactions in Operational Research* **7**(6), 637 – 651.

[3] G. Dudek and H. Stadtler (2005), Negotiation-based collaborative planning between supply chains partners. *European Journal of Operational Research*, **163**, 668 – 687.

[4] D.E. Goldberg (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley.

[5] Y. Hendel, F. Sourd (2006), An improved earliness–tardiness timing algorithm. *Computers & Operations Research*, In Press.

[6] J.A. Hoogeven, S.L. Van de Velde (2001), Scheduling with target start times. *European Journal of Operational Research*, **129**, 87 – 94.

[7] S. Hurtubise, C. Olivier, A. Gharbi, Planning tools for managing the supply chain (2004) Computers & Industrial Engineering, **46**, 763 – 779.

[8] L. Ouzizi, D. Anciaux, M.-C. Portmann, F. Vernadat (2006), A model for co-operative planning within a virtual enterprise. *International journal of Computer Integrated Manufacturing*. **19**(3), 248 – 263.

[9] M.-C. Portmann (1988), Méthodes de décompositions spatiales et temporelles en ordonnancement de la production. *RAIRO-APII*, **22**(5), 439 – 451.

[10] M.-C. Portmann, A. Vignier (2000) Performances' study on crossover operators keeping good schemata for some scheduling problems, Proceedings of GECCO'2000, 331 – 338.

[11] F. Sourd (2005), Punctuality and idleness in just-in-time scheduling. *European Journal of Operational Research*, **167**(3), 739 – 751.

Adaptive Decomposition and Construction for Examination Timetabling Problems

Rong Qu, Edmund K. Burke
Automated Scheduling, Optimisation and Planning (ASAP) Group, School of CSiT
University of Nottingham, Nottingham, NG8 1BB, UK, {rxq, ekb}@cs.nott.ac.uk

Decomposition techniques have not been widely investigated in timetabling research mainly due to the complexity of the problems. In this paper, we develop a new general adaptive decomposition technique where problems are iteratively partitioned into two sub-sets, each containing a set of events with different levels of difficulty. The events in these two sets, namely the *difficult set* and the *easy set*, are ordered in turn and used to construct the solutions. Potentially difficult events are adaptively included in the *difficult set*, whose size is also adaptively adjusted according to the solution quality obtained in previous iterations. It is observed that in most cases, the *difficult set*, although of small size, contributes to a much larger portion of the total cost of the solutions constructed. This simple yet effective adaptive technique obtained competitive solutions compared with state-of-the-art approaches in the literature for benchmark exam timetabling problems.

Keywords: Adaptive, Decomposition, Timetabling.

1 Introduction

As one of the most important administrative activities in all universities, the exam timetabling process has attracted significant research attention in the last four decades [**?**]. A general exam timetabling problem consists of scheduling a list of exams into a limited set of timeslots so as to satisfy some constraints absolutely (the *hard* constraints) and some others as much as is possible (the *soft* constraints). The most common hard constraint is that of avoiding assigning exams with common students to the same timeslot. The most common soft constraint in the literature is to spread students' exams as far as possible to allow enough revision time. These constraints are considered in the benchmark exam timetabling problems tested in this paper. They were introduced by Carter, Laporte and Lee [9] in 1996 and have since been widely used in the literature. The problem consists of assigning exams (81-682 across different instances) to a limited number of timeslots, while satisfying the above hard and soft constraints. The quality of the solutions is evaluated by the average cost of how exams are spread for each students. During the years, there has been an issue with different instances circulating under the same name. This situation is clarified in [15]. More details can be found at http://www.cs.nott.ac.uk/∼rxq/data.htm.

The last ten years have seen a considerable increase in the number of research publications where meta-heuristics were developed for educational timetabling (e.g. [7, 15, 16]). These include Tabu Search (e.g. [14, 18]), Simulated Annealing (e.g. [12, 17]) and Evolutionary Algorithms (e.g. [6]), etc. Usually different mechanisms are defined to underpin efficient exploration of the search space. Very often, the methodologies in the literature represent tailor-made algorithms that work very well on the specific problem for which they developed but not on others.

Recent research in timetabling has seen some development on improving the flexibility and generality of search algorithms (see [15]). For example, variable neighbourhood search [10] and large scale neighbourhood search [1] employ different neighborhood structures to enable wider exploration in the search space and have been developed with some success for exam timetabling.

Another example is provided by hyper-heuristic research which search upon a space of heuristics rather than upon the actual solutions (e.g. [3]) in attempt to be more generally applicable across a wider range of problems. Examples of recent research papers on hyper-heuristic approaches to exam timetabling include [2, 5].

This paper explores adaptive techniques, which are relatively new in timetabling. In [4], a methodology was developed to adaptively order the exams by how difficult they were in the previous solution's construction. At each iteration, those exams which contributed to costs greater than a certain threshold were assigned increased difficulty values and were ordered and scheduled earlier to construct solutions. This simple adaptive approach was very efficient on the benchmark problems tested. It was based upon the "squeaky wheel" optimisation technique [11] which was originally applied to both scheduling and graph coloring problems.

The basic idea of decomposition is to "divide and conquer", as (near) optimal solutions may be obtained more easily for smaller sub-problems using relatively simple approaches [8]. However, the task of decomposing the problem is challenging and problem specific. Determining how the sub-solutions obtained can be combined for the original problem also represents a key issue. There are very few papers in the timetabling literature which have investigated decomposition techniques. In [6], a look ahead mechanism in a multi-stage approach considered two sub-problems at a time. The methodology of [6] significantly reduced the computational time and improved the solution quality on the addressed exam timetabling benchmark. In [9], a clique of the graph that modelled the timetabling problems is first obtained and used to generate solutions. This can be seen as decomposing the problems into two parts, which are used one after another to construct solutions.

2 The Adaptive Decomposition and Construction Approach

In our adaptive decomposition approach, an initial ordering of the exams is first obtained by the Saturation Degree graph heuristic (which orders the exams by the number of remaining feasible timeslots during the solution construction). In some cases, the ordering may need to be adjusted by randomly swapping two exams in the ordering until a feasible solution can be obtained. Based on this initial ordering, this list of exams is decomposed adaptively into two subsets (called the *difficult* and the *easy set*) by a two-stage approach, which is described next.

In a constructive approach, if the assignment method that schedules exams into the timeslots is fixed, then the problem can be seen as being transferred into an ordering (permutation) problem. This problem has a search space of size $e!$, (where e is the number of exams) and thus is much larger than that of the original problems (of size t^e, where t is the number of timeslots). The adaptive decomposition approach decomposes the original ordering problem into two smaller ordering sub-problems and thus the size of the search space is significantly reduced.

2.1 Adaptive Detection on the *Difficult Set*

In the first stage, both the exams and their ordering in the *difficult set* are adaptively adjusted by using information derived from the solution construction that was observed in previous iterations. It is an iterative process where, at each iteration, the exams in both the *difficult set* and the *easy set* are used to construct a complete solution, whose quality is used to identify problematic exams. The aim is to collect these troublesome exams into the *difficult set*, which can be dealt with with higher priority in future iterations of solution construction. Thus better solutions can be generated.

The pseudocode presented in *Algorithm 1* outlines the process. In each iteration, ordering of the exams in the *difficult set* is adjusted to find the best order that produces an improved or feasible solution. If no feasible solution can be generated using the current ordering, then the exam which

cannot be scheduled is moved forward in the *difficult set* ordering, and the *difficult set* is reduced to include only the exams before this exam. Initial tests showed that moving the exam 2 to 6 positions forward obtained good results thus 5 is selected in the approach. If a feasible or improved solution is obtained, then this order of exams is kept and the *difficult set* is increased to include the first exam in the *easy set* to detect more potentially difficult exams iteratively. The optional steps in *Algorithm 1* are only taken in the experiment presented in Section 3.2 for comparison purpose.

Algorithm 1: DIFFICULT_SET_DETECTION()

build the initial ordering of exams by Saturation Degree
initial size of difficult set Sd = number of exams / 2
MaxNoIterations = 10,000; iteration = 0
while iteration<MaxNoIterations
do
- easy set = $\{e_{Sd+1}, e_{Sd+2}, \ldots e_e\}$
- reorder the exams in the difficult set $\{e_1, e_2, \ldots e_{Sd}\}$
- construct a solution using ordered exams in both difficult and easy sets
- //optional step: calculate costs of exams in the constructed solution
- //optional step: move forward the exam incurring the highest cost
- **if** a feasible solution or an improved solution is obtained
  - Sd = Sd + 1 // include more potential exams in the difficult set
- **else**
  - move forward the difficult exam causing the infeasibility
- store the difficult set and its size S_{best} if the best solution is obtained
- iteration = iteration + 1

2.2 Ordering in the *Easy Set*

In the second stage, after the adaptive decomposition, the potentially difficult exams and their ordering in the *difficult set* are fixed. The exams in the *easy set* are then reordered to further improve the quality of solutions built by scheduling the ordered exams one by one in both the *difficult set* and the *easy set*.

In most decomposition approaches in the literature [6, 8], the sub-solutions obtained need to be carefully combined together. This particularly concerns global information pertaining to the original problem. In our adaptive decomposition approach, a solution is constructed by ordering the exams in one set while keeping the other set fixed. It thus deals with the global problem information as both sets are concerned while solving the sub-problems, thus no adaptation needs to be made. Also the *difficulty* of exams is adaptively obtained (rather than by using a fixed measure) based on online information from the previous solution quality value for the problem in-hand.

3 Experimental Results on the Benchmark Data Set

We carried out a set of experiments to analyse the adaptive decomposition and construction approach. We were particularly concerned with the effects of the decomposition, the ways the *difficult set* is adjusted, the relationship between the two sets, and the effects that the *difficult set* has on the overall solution quality. They are presented in the next sub-sections, respectively. For each experiment, five runs were carried out, from which both the average and best results are reported to give a better evaluation of the approaches. All the approaches in different experiments were run

for the same number of iterations for fair comparisons. The results obtained by our approach, when comparing with the state-of-the-art approaches, indicate the efficiency, simplity and generality of the adaptive decomposition approach (see section 3.5). The coding is in C++ and experiments were carried out on a PentiumIV 3GHZ machine with 1G memory. Computational time is reported here only for comparisons between variants of the adaptive decomposition approach. Most of the approaches in the literature did not report the computational time as it is impossible to compare it across different platforms. Also, time is not a crucial issue in real world circumstances as usually the timetables are built weeks or months before they are utilised. The version of the exam timetabling problem that we tackle is that introduced in [9] and the naming conventions for the datasets are those introduced in [15].

3.1 Analysis on the Adaptive Decomposition

In the first set of experiments, the effect of decomposition in the adaptive approach is evaluated. We developed an adaptive approach without decomposition. That is, a single set of ordered exams is adjusted adaptively to construct solutions. At each iteration, the exam causing infeasibility is moved forward. Note that this approach is different from the adaptive ordering approach developed in [4], where at each iteration a subset of exams with costs higher than a certain threshold are given a higher priority in the next iteration. Different thresholds were thus tested in [4]. In this study, we try to keep the approach simple by introducing the least number of parameters.

This approach is compared with the adaptive decomposition approach presented in Section 2 and results are given in Table 1. It can be seen that the adaptive decomposition approach performs significantly better than its variant without decomposition. On all of the 11 problems, the improvement on solution quality ranges from 0.69% - 11.02%. This indicates that the adaptive decomposition can effectively decompose the problems in hand and produce better results. The adaptive approach without decomposition takes slightly more time.

Table 1: Average and best results from the adaptive approach with (upper I) and without (lower II) decomposition on benchmarks. (the best and average results are highlighted in bold; improve % = (without-with)/with

	car91 I	car92 I	ear83 I	hec92 I	kfu93 I	lse91 I	sta83 I	tre92 I	ute92 I	uta93 I	yor83 I
I avg	5.47	4.7	39.14	12.21	15.43	11.66	162.44	9.18	28.03	3.6	45.1
best	**5.38**	**4.53**	**36.76**	**11.45**	**14.79**	**11.2**	**157.4**	**8.83**	**26.87**	**3.53**	**42.04**
time (s)	3104	2140	126	32	185	148	47	192	39	2736	126
II avg	5.55	4.66	41.7	12.53	16.86	12.26	158.6	9.17	28.43	3.71	43.84
best	5.53	4.59	41.7	12.26	16.2	12.05	157.77	8.89	27.81	3.68	42.12
time (s)	2993	2020	128	32	191	180	32	183	30	2616	110
improve %	1.46	1.57	11.02	2.95	11.01	7.92	0.69	0.99	2.52	2.9	0.93

3.2 Analysis on the Construction of the *Difficult Set*

In the second set of experiments, a variant of the adaptive decomposition approach was developed. Not only is the exam (from the *difficult set*) that incurs an infeasibility moved forward. Here, we implement the optional steps in *Algorithm 1* to see what benefit this could have in the adaptive decomposition approach. The above two ways of adjusting the *difficult set* are compared and presented in Table 2. The results obtained in both the 1^{st} stage (by detecting the *difficult set*) and the 2^{nd} stage (by ordering the *easy set*) are also presented. The computational time for the adaptive decomposition is the same as that in Table 1 so is not presented again in Table 2.

Table 2: Average and best results from the adaptive approach forwarding only exams causing infeasibility (upper "I") and also exams incurring the highest cost (lower "III"). "1^{st}" and "2^{nd}" present the results obtained after stage 1 and stage 2, respectively. "improve %" = (1^{st}-2^{nd})/1^{st}

	car91 I	car92 I	ear83 I	hec92 I	kfu93 I	lse91 I	sta83 I	tre92 I	ute92 I	uta93 I	yor83 I
I 1^{st} avg	5.59	4.7	39.54	12.21	15.43	11.66	162.44	9.18	28.03	3.67	45.1
2^{nd} avg	5.47	4.68	38.43	12.18	15.19	11.36	157.51	9.08	27.73	3.6	43.1
improve %	2.2	2.4	2.8	2	1.6	2.6	3	1.1	1	1.9	3.7
best	5.38	**4.53**	**36.76**	**11.45**	14.79	11.2	**157.4**	8.83	**26.87**	3.53	**42.04**
III 1^{st} avg	5.5	4.69	41.2	12.21	15.5	11.5	158.85	9.08	27.95	3.68	45.25
2^{nd} avg	5.38	4.59	40.66	12.2	15.3	11.31	157.67	9.01	27.72	3.62	43.72
improve %	2.2	2.3	1.3	0.1	1.3	1.7	0.7	0.8	0.8	1.58	3.4
best	**5.32**	**4.53**	38.59	11.75	**14.63**	**10.96**	157.51	**8.81**	26.88	**3.51**	42.16
time (s)	3203	2198	138	36	185	148	42	203	36	2980	154

We can see that these two approaches perform quite similarly on the benchmark problems (i.e. they obtained better results on 5 problems, respectively, and the same results on 1 problem). This indicates that considering exams of the highest cost does not improve the approach. Also, more computational time is required in the variant as, at each iteration, the costs of exams need to be re-calculated. It can also be observed that in both of the variants, reordering the *easy* exams also contributes to better solutions based on the *difficult set* obtained from the 1^{st} stage.

3.3 Analysis on the *Difficult Set* vs. the *Easy Set*

We further investigate the relationship between the *difficult set* and the *easy set* in the adaptive decomposition approach. Instead of using a distinct boundary between the two sets, in the 2^{nd} stage, the *easy set* will also include the 2^{nd} half of the *difficult set*. That is, the *difficult set* and the *easy set* have some overlapping exams (i.e. the exams in $\{e_{S_{best}/2}, e_{S_{best}/2+1}, \ldots e_{S_{best}}\}$ are also considered in the 2^{nd} stage, S_{best} is obtained from *Algorithm 1*).

Table 3: Average and best results from the adaptive decomposition approach with (lower *IV*) and without (upper *I*) and overlapping between the *difficult set* and the *easy set*.

	car91 I	car92 I	ear83 I	hec92 I	kfu93 I	lse91 I	sta83 I	tre92 I	ute92 I	uta93 I	yor83 I
I avg	5.47	4.7	39.14	12.21	15.43	11.66	162.44	9.18	28.03	3.6	45.1
best	**5.38**	4.53	36.76	11.45	14.79	11.2	157.4	8.83	26.87	**3.53**	**42.04**
IV avg	5.5	4.58	37.85	12.09	15.16	11.31	157.55	8.98	27.6	3.61	43.27
best	5.45	**4.5**	**36.15**	**11.38**	**14.74**	**10.85**	**157.21**	**8.79**	**26.68**	3.55	42.2

It can be clearly seen from Table 3 that considering overlapping exams in the *easy set* contributes to a better performance for the approach. It obtained better solutions on 8 out of 11 problems. This indicates that some exams could be considered as both *easy* and *difficult* in the problem, and there is no distinct boundary between the two subsets. Computational time is almost the same for all the problems (thus it is not presented in Table 3).

3.4 Contributions of the *Difficult Set* to the Overall Solution Quality

Based on the above experiments, we finally analyse the exams in the *difficult set* detected by the above different adaptive approaches and evaluate their contribution to the overall cost of the solutions generated. Table 4 presents these evaluations.

Table 4: Average size ("size %") of the *difficult set* in different adaptive approaches. (I: forward exams causing infeasibility; III: I + forward exams of the highest cost; IV: I + overlapping *difficult set* and *easy set*. "match %" presents the number of exams in the *difficult set* that matches the first 50% of exams with the highest cost. "cost %" presents the summed cost occurring from the exams in the *difficult set*.

	car91 I	car92 I	ear83 I	hec92 I	kfu93 I	lse91 I	sta83 I	tre92 I	ute92 I	uta93 I	yor83 I
size %	32	23	38	61	23	14	15	46	37	22	44
I match %	89	88	83	73	99	98	32	86	66	98	68
cost %	66	62	60	71	83	58	12	79	57	66	47
size %	33	25	44	70	27	18	43	47	24	51	43
III match %	88	91	75	65	99	94	40	84	97	64	65
cost %	66	64	64	72	87	65	30	80	67	67	43
size %	32	23	38	61	23	14	15	46	37	22	44
IV match %	87	87	79	71	91	95	49	83	63	98	63
cost %	65	62	59	70	78	58	23	79	54	66	46

First, we obtain the size of the *difficult set* with respect to the overall size of the original problem. It can be seen that, in all cases except "hec92 I", less than half of the exams are detected as *difficult* in the problem. The problems with the largest *difficult set* are "hec92 I", "tre92 I" and "yor83 I", which are actually the most difficult problems, in the sense that a feasible solution cannot be obtained using a pure Saturation Degree. It is also observed that the sizes of the detected *difficult set* by these different adaptive approaches are consistent, indicating that these approaches are adapting appropriately to very different problems.

To evaluate how important the exams are in the *difficult set*, we first order (in a descending manner) all the exams $e_1 > e_2 > ... > e_e$ by their costs in the best solution generated. Then the first 50% of the exams with the highest costs are checked to see if they are also detected as difficult exams in the *difficult set* (i.e. we check the membership of $\{e_1, e_2...e_{e/2}\}$). It can be seen that in all the problems, up to 98% of the difficult exams are contributing to the highest costs in the solutions. The exception is sta83 I, where only 12% - 30% of the difficult exams are incurring the highest costs.

Whilst the above evaluation quantitively indicates the difficulty of the exams in the *difficult set*, we further evaluate qualitively the difficulty of these exams. We calculated the summed cost incurred from the exams in the *difficult set* to the total cost of the solutions. In most cases, the cost from the *difficult set* contributes to 54% - 87% of the total cost except in the case of "sta83 I" and "yor83 I". Note that this should be read in conjunction with the size of the *difficult set*, which, in most cases, is much smaller, indicating that in most problems, ***a small set of difficult exams contributes to a large portion of the cost of the solutions***.

It is interesting to see that for problems "hec92 I", "sta83 I" and "yor83 I", roughly the same proportion of difficult exams contributes to the same proportion of cost and the difficult set (i.e. percentages across "size %", "match %" and "cost %" are close to each other for each problem). This indicates that almost all the exams in these problems are equally important. Furthermore, it also explains why forwarding the highest costs in approach III does not contribute to better performance of the adaptive approach. Further study on these problems in comparison with the others may reveal other interesting observations.

3.5 Comparisons with State-of-the-art Approaches

During the years, a number of approaches have been developed to solve these benchmark exam timetabling problems. We present the best results (taken from [15]) in Table 5 and evaluate our

adaptive decomposition method (ADA) against these best results and against the adaptive method of [4].

Table 5: Best results from the adaptive decomposition approach with overlapping exams (ADA) and the best reported from the state-of-the-art approaches on benchmark exam timetabling problems.

	car91 I	car92 I	ear83 I	hec92 I	kfu93 I	lse91 I	sta83 I	tre92 I	ute92 I	uta93 I	yor83 I
ADA	5.45	4.5	36.15	11.38	14.74	10.85	**157.21**	8.79	26.68	3.55	42.2
[4]	4.97-5.55	4.32-4.68	36.16-38.55	11.61-12.82	15.02-16.5	10.96-12.53	161.91-170.53	8.38-8.96	27.41-29.67	3.36-3.6	40.77-42.97
best	4.2	4.0	29.3	9.2	13.46	9.6	157.3	8.13	24.21	3.2	36.11

By comparing our adaptive decomposition approach with that of the adaptive ordering approach developed in [4], we can see that in the 11 problems, our approach obtained better results on 5 problems, and similar results (i.e. within the range of results from [4]) for the other problems. Note that in [4] a number of variants were tested by adjusting the difficulty values added to the measure of exams in different ways. Also a threshold needs to be chosen to decide which exams need to be moved forward. Furthermore, in each iteration, the difficulty of all of the exams needs to be re-calculated, based on which an ordering can be set. Our approach does not involve these calculations on each exam in each iteration and thus is much simpler.

For the benchmark problems, different approaches in the literature worked particularly well on specific instances (but not on all of them). These best results are summarised in [15]. Compared with all the other approaches, our adaptive decomposition approach obtained competitive results which are in the range of the best reported. In particular, for problem "sta83 I", our approach obtained the best result (157.21) reported in the literature. Note that our approach is a simple pure constructive approach and does not reply on initial solutions and fine tuning of a number of parameters, which are crucial to the success of most approaches.

4 Conclusions and Future Work

This paper develops an adaptive decomposition and ordering approach which constructs solutions for exam timetabling problems. The original problems are decomposed adaptively into two subsets, the *difficult set* and the *easy set*. They are used to construct solutions by adjusting the orderings of the exams in one set while fixing the other. The approach integrates both the adaptive and the decomposition techniques and has links to a number of approaches developed in the field. Our first aim is to decompose the complex timetabling problems into small problems which are easier to handle. Another aim is to automatically integrate adaptive mechanisms into the simple constructive technique when dealing with different problems. Current non-adaptive mechanisms are usually specially designed and hard coded in most of the meta-heuristic approaches in the literature [15].

It is observed by experimentation that the potentially difficult exams detected by our approach usually represent a small proportion of the original problems but contribute to a large proportion of the costs of the solutions. The difficulty of the exams is adaptively adjusted during the problem solving rather than using static or pre-defined methods. It is observed that ordering the exams in the *easy set* also contributed to the generation of better solutions.

The comparisons of this adaptive decomposition approach with the state-of-the-art approaches indicate that it is a simple yet effective technique. It is also a general technique which can be adapted to quickly construct good quality solutions to any problems which can be solved using heuristic ordering strategies.

There are also exceptions detected by the adaptive decomposition approach i.e. that, in some cases, almost all exams are equally difficult when ordered and used to construct solutions. This requires further study. Also, the *difficult set* and the *easy set* do not have a distinct boundary. Future work will study more learning and classification techniques for detecting the *difficult set* more accurately. It will also be interesting to study more elaborate ordering methods on both the *difficult set* and the *easy set*.

References

[1] R.K. Ahuja, J.B. Orlin and D. Sharma (2000), Very Large Scale Neighbourhood Search, *International Transactions in Operational Research* **7**, 301 – 317.

[2] B. Bilgin, E. Ozcan and E.E. Korkmaz (2006), An experimental study on hyper-heuristics and exam timetabling, Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling, 123 – 140.

[3] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross and S. Schulenburg (2003), Hyper-heuristics: an Emerging Direction in Modern Search Technology, In: Glover F. and Kochenberger G. (eds), *Handbook of Meta-Heuristics*, Kluwer, 457 – 474.

[4] E.K. Burke and J. Newall (2004), Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings, *Annals of Operational Research* **129**, 107 – 134.

[5] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic and R. Qu (2007), A Graph-Based Hyper Heuristic for Timetabling Problems, *European Journal of Operational Research* **176**, 177 – 192.

[6] E.K. Burke and J. Newall (1999), A Multi-Stage Evolutionary Algorithm for the Timetabling Problem, *The IEEE Transactions on Evolutionary Computation* **3**(1), 63 – 74.

[7] E.K. Burke and S. Petrovic (2002), Recent research directions in automated timetabling, *European Journal of Operational Research* **140**(2), 266 – 280.

[8] M.W. Carter (1983), A decomposition algorithm for practical timetabling problems., Technical Paper 83-06, Department of Industrial Engineering, University of Toronto.

[9] M. Carter, G. Laporte and S. Lee (1996), Examination Timetabling: Algorithmic Strategies and Applications, *Journal of Operational Research Society* **47**, 373 – 383.

[10] P. Hansen and N. Mladenovic (2001), Variable Neighbourhood Search: Principles and Applications, *European Journal of Operational Research* **130**, 449 – 467.

[11] D.E. Joslin and D.P. Clements (1999), "Squeaky Wheel" Optimization, *Journal of AI Research* **10**, 353 – 373.

[12] L. Merlot, N. Boland, B. Hughes and P. Stuckey (2002), A Hybrid Algorithm for the Examination Timetabling Problem. In: Selected Papers from PATAT02, Lecture Notes in Computer Science **2740**, 207 – 231.

[13] B.G.C. McCollum (2006), Bridging the Gap between Research and Practice, Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling, (Keynote) 15-35. August 2006, Czech Republic.

[14] K. Nonobe, T. Ibaraki (1998), A Tabu Search Approach to the Constraint Satisfaction Problem as a General Problem Solver, *European Journal of Operational Research* **106**, 599 – 623.

[15] R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot and S.Y. Lee (2006), A Survey of Search Methodologies and Automated Approaches for Examination Timetabling, Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham.

[16] A. Schaerf (1999), A Survey of Automated Timetabling, *Artificial Intelligence Review* **13**(2), 87 – 127.

[17] J. Thompson and K. Dowsland (1998), A Robust Simulated Annealing Based Examination Timetabling System. *Computers & Operations Research* **25**, 637 – 648.

[18] G.M. White, B.S. Xie B.S. and S. Zonjic (2004), Using Tabu Search with Longer Term Memory and Relaxation to Create Examination Timetables, *European Journal of Operational Research* **153**(16), 80 – 91.

Aligning Frequencies in Cyclic Delivery Scheduling

Birger Raa and El-Houssaine Aghezzaf
Department of Industrial Management, Ghent University, Technologiepark 903, 9052 Zwijnaarde, Belgium,
birger.raa@UGent.be

When facing the task of replenishing customers with stable demand rates on a long-term basis, the natural approach is to set up a cyclic distribution pattern. In this cyclic approach, customers are grouped into tours and tours are assigned to the available vehicle fleet. To ensure that a vehicle can feasibly make all tours that are assigned to it, the cycle times of these tours need to be aligned. This paper presents a new heuristic that aligns delivery frequencies of multiple tours that have to be made by a single vehicle, with the objective of minimizing cost rates. Computational experiments are set up to evaluate the performance of the new heuristic and compare it to an existing heuristic.

Keywords: Delivery Scheduling, Transport Scheduling, Heuristic Search.

1 Problem description

In many real-life distribution systems, a set of customers has to be repeatedly replenished from a depot. When these customers have stable consumption rates, the natural approach is to set up a cyclic scheme. In such a cyclic approach, customers are grouped into tours and then the best cycle time for each tour is determined. If the holding costs of the customers are taken into account, this best cycle time can be determined by an EOQ-like formula [4], that trades off transportation and holding costs. However, the problem of setting up a cyclic distribution scheme has a long-term perspective. Therefore, it is appropriate not to consider the vehicle fleet as fixed. Indeed, in the long term, the vehicle fleet is variable, such that fleet sizing needs to be taken into account. This means that the tours have to be assigned to a minimal fleet of vehicles. Campbell and Hardin [1] propose a heuristic for this problem. But, since fixed vehicle costs are usually large compared to transportation and holding costs, it may be a good idea to adjust the cycle times of one or more tours in an attempt to reduce the required number of vehicles. This article studies the problem of aligning the cycle times of tours that are assigned to a single vehicle, such that a feasible schedule can be constructed for the vehicle. Consider e.g. the two tours of Figure 1 that each take a full working day to be completed. If the optimal cycle times of these tours are two and three days respectively, then they will have to be performed on the same day once every six days. Since they each take a full day to be performed, two vehicles would be needed. However, if we align the cycle times of these tours and decide to decrease the cycle time of the second tour to two days or increase it to four days, both tours never have to be made on the same day and only one vehicle is needed. Adjusting the cycle time of the second tour to 2 days as in Figure 1 increases the variable distribution costs and decreases holding costs, thus disturbing the trade-off between these, but it helps in reducing the fixed vehicle fleet costs, so it is certainly worthwhile.

When adjusting tour cycle times, we need to know the range of feasible cycle times for each route individually. It is assumed here that no tour is ever made twice per day, so the obvious minimal cycle time for any given tour is 1 day. The maximal tour cycle time, on the other hand, is derived from the limited loading capacity of the vehicles and the limited storage capacity of the customers. With κ the vehicle loading capacity, S the subset of customers visited in the tour, d_j

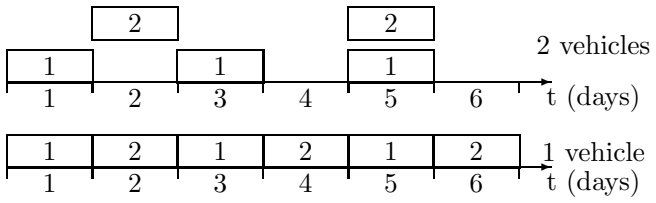


Figure 1: A simple example of cycle time alignment

the daily demand rate of customer j and κ_j the storage capacity of customer j, the maximal cycle time of the tour is as follows: $T_{max} = \min\left(\frac{\kappa}{\sum_{j\in S} d_j}, \min_{j\in S} \frac{\kappa_j}{d_j}\right)$.

The optimal cycle time gives the best trade-off between transportation and holding costs. Since holding costs are usually small compared to transportation costs, large delivery quantities and thus long cycle times give better trade-offs. The size of this delivery quantity is limited by the available capacities and thus the optimal cycle time is very often given by the maximal cycle time.

Formal problem description

Given a set of N different tours (indexed by i) with their optimal cycle times $T_{eoq,i}$ in days, their maximal cycle times $T_{max,i}$ in days and their travel times t_i in hours, we need to find tour cycle times T_i and a delivery schedule for a single vehicle in which tour i is made once every T_i days and in which the cumulative travel time of the vehicle on any given day does not exceed a given threshold (usually 8 hours). The objective is to minimize the variable distribution and holding cost rate. If no cost information is explicitly given, this objective can be replaced by the minimization of the cumulative deviation between the actual cycle times T_i and the optimal cycle times $T_{eoq,i}$.

A mathematical formulation for this problem is given below. The variable T in the model denotes the cycle time of the vehicle's schedule. This schedule cycle time T is given by the least common multiple of all tour cycle times T_i. The binary variables X_{it} indicate whether tour i is made on day t in the schedule or not.

$$\text{Min Z} = \sum_{i=1}^{N} \mathbf{abs}\left(1 - \frac{T_i}{T_{eoq,i}}\right)$$

subject to:

$$\begin{aligned}
\sum_{t=1}^{T_i} X_{it} &= 1 && \forall i \in 1..N \\
X_{it} &= X_{i,t+T_i} && \forall i \in 1..N, t \in 1..T - T_i \\
\sum_{i=1}^{N} t_i X_{it} &\leq 8 && \forall t \in 1..T \\
T_i &\in \{1..T_{max,i}\} && \forall i \in 1..N \\
X_{it} &\in \{0,1\} && \forall i \in 1..N, t \in 1..T
\end{aligned}$$

Because the cycle times T_i (and T) are variables, this formulation is non-linear. When the T_i and T are fixed, the problem becomes a linear generalized assignment problem, which is known to be NP-hard. When the number of tours assigned to a single vehicle is limited, the problem can be solved by enumerating all possible T_i combinations and solving the resulting linear MIP-model

with branch-and-bound. However, this is very time-consuming. To be able to solve larger instances and to be able to evaluate a large number of solutions for the overall problem of setting up a cyclic distribution scheme with many tours and multiple vehicles, a fast heuristic is developed for the cycle time alignment (sub)problem.

2 Solution approach

The heuristic approach for the cycle time alignment problem consists of two iterative phases: (i) selecting tour cycle times T_i, starting of course from the optimal cycle times $T_{eoq,i}$ and (ii) checking the feasibility of a given T_i combination by constructing a schedule. As soon as a feasible schedule is found, the procedure stops and the cycle time combination is returned.

The cycle time T, given by the least common multiple of the cycle times T_i, gives an indication of the 'alignment' of the cycle times. A high T indicates that some of the tour cycle times are relative primes, meaning that the tours have to be made on the same day every now and then. To avoid having to make the tours on the same day, their cycle times have to be aligned such that they are no longer relative primes. This increases the 'alignment' and reduces the cycle time T.

During the alignment procedure, tours are sorted in order of increasing cycle times T_i, and in order of decreasing durations t_i in case of a tie. Because the relative cycle time deviations are to be minimized, the idea is to adjust the larger T_i's first. To this end, a critical tour is identified in the second phase, i.e. during schedule construction. In the first phase, the cycle times of all tours in the order starting from this critical tour are adjusted as follows. First, the least common multiple of all tours, which is the cycle time T of these tours together, before the critical tour is calculated. Then, the cycle time of the tour is first decreased and later increased until it is a divisor or an integer multiple of T. Of these two, the one with smallest deviation from the tour optimal cycle time is chosen.

The second phase, where a feasible schedule has to be constructed for given cycle times T_i, consists of two steps. The first step is a preprocessing step that compares the T_i two by two. If two tours have relative prime cycle times and cannot be made together on the same day without violating the driving time restriction, we can already conclude here that no feasible schedule exists. The tour with the higher T_i then becomes the critical tour and the procedure returns to the first phase. When no conflicts are encountered in the first step, the second step is started, in which an actual schedule is constructed with a best-fit insertion heuristic, adapted from Raa et al. [2]. In this heuristic, the tours are inserted into the schedule one by one according to the given tour order, except that, if possible, no two tours with the same cycle time are inserted consecutively [2]. As soon as a tour cannot be feasibly inserted, it becomes the critical tour and the procedure returns to the first phase of cycle time alignment. If a feasible schedule is found, the problem is solved and the procedure stops.

To make this approach work, two extra rules are added. The first is that if the critical tour is the same as in the previous iteration, the tour before it in the order becomes critical. This is necessary to avoid infinite looping. The second rule looks at the vehicle utilization. If the total travel time during a cycle is very high compared to the time available in the cycle, this greatly increases the chances of resulting in infeasibilities. Therefore, when the vehicle utilization is too high, only cycle time increases are allowed in the first phase so the vehicle utilization will decrease.

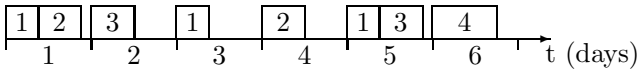
3 Illustrative example

i	$T_{eoq,i}$	$T_{max,i}$	t_i
1	2	4	3h
2	3	4	4h
3	4	6	4h
4	7	7	6h

Initially, the T_i are equal to the optimal cycle times $T_{eoq,i}$. This gives a (very high) cycle time T of 84 days. However, tours 1 and 4 conflict: they have relative prime cycle times and take more than 8 hours together. Therefore, tour 4 becomes critical and its cycle time T_4 is adjusted to 6 days (increasing to 8 is infeasible because $T_{max,4} = 7$), reducing T to only 12 days.

With the adjusted T_4, no conflicts can be found in the first step of the second phase, so a schedule is constructed. This schedule construction fails when inserting tour 4, but since this was already critical, tour 3 becomes critical and its cycle time T_3 is adjusted to 3 days (increasing to 6 would result in a larger deviation). The cycle time T_4 remains unchanged because it is already aligned. This results in the feasible schedule shown below.

i	T_i	$1 - \frac{T_i}{T_{eoq,i}}$
1	2	0.0%
2	3	0.0%
3	3	25.0%
4	6	14.3%



Computational results

To evaluate the performance of the proposed procedure for cycle time alignment, it is tested on a set of randomly generated instances and then compared to the results obtained with the approach of Raa and Aghezzaf [3, 4]. That approach is based on relative frequencies, and iteratively determines a cycle time T and a set of frequencies k_i for the tours, such that the resulting tour cycle times T/k_i are as close as possible to the optimal tour cycle times. The heuristic used for constructing the schedule is the same as the one used here. These computational experiments are currently being performed and results will be available soon.

4 Conclusion

This article presents a new heuristic solution approach for finding cycle times for tours that are assigned to a vehicle, such that the cumulative deviation from their individual optimal cycle times is minimal. The approach is very fast and effective, which is particularly useful when designing cyclic distribution patterns for replenishing large sets of customers, where many tours and vehicles are necessary and thus many alternatives need to be evaluated. Currently, computational experiments are being performed to evaluate the effectiveness of this new approach.

For future work, the proposed heuristic will need to be adapted to tackle variants of the problem with additional real-life features and constraints such as heterogenous vehicle fleets and customer time windows for delivery.

References

[1] A.M. Campbell and J.R.Hardin (2005), Vehicle minimization for periodic deliveries, *European Journal of Operational Research* **165**, 668 – 684.

[2] B. Raa, E.-H. Aghezzaf and W. Dullaert (2006), Cyclic Scheduling of Multiple Tours with Multiple Frequencies for a Single Vehicle, submitted to the *International Journal of Logistics Systems and Management*.

[3] B. Raa and E.-H. Aghezzaf (2006), A practical solution approach for the cyclic inventory routing problem, submitted to the *European Journal of Operational Research*.

[4] B. Raa (2006), Models and algorithms for the cyclic inventory routing problem, Ph.D. thesis, Ghent University.

Scheduling Single Round Robin Tournaments with Fixed Venues

Rafael A. Melo
Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil, melo@ic.uff.br
Sebastián Urrutia
Department of Computer Science, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-010, Brazil, surrutia@dcc.ufmg.br
Celso C. Ribeiro
Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil, celso@inf.puc-rio.br

Sports scheduling is a very attractive application area due to the importance of the problems in practice and to their interesting mathematical structure. We introduce a new problem with practical applications, consisting in scheduling a compact single round-robin tournament with fixed venue assignments for each game. Two integer programming formulations are proposed and compared. Comparative numerical results are presented.

Keywords: Sports scheduling, timetabling, integer programming.

1 Problem statement

The field of sports scheduling has been attracting the attention of an increasing number of researchers in multidisciplinary areas such as operations research, scheduling theory, constraint programming, graph theory, combinatorial optimization, and applied mathematics. Particular importance is given to round robin scheduling problems in which each team is associated with a particular venue, due to their relevance in practice and to their interesting mathematical structure. The difficulty of the problems in the field leads to the use of a number of approaches, including integer programming [10, 15], constraint programming [6], hybrid methods [3], and heuristic techniques [1, 14]. We refer to [4, 11] for literature surveys.

In round robin tournaments, every team face each other a fixed number of times in a given number of rounds. Every team face each other exactly once in a single round robin (SRR) and twice in a double round robin (DRR). If the number of rounds is minimum and every team plays exactly one game in every round, then the tournament is said to be compact. Each team has its own venue at its home city. Each game is played at the venue of one of the two teams in confrontation. The team that plays at its own venue is called the home team and is said to play a home game, while the other is called the away team and is said to play an away game. A schedule must not only determine in which round each game will be played, but also at which venue.

The problem of scheduling round robin tournaments is often divided into two subproblems. The construction of the timetable consists in determining the round in which each game will be played. The home-away pattern (HAP) set determines in which condition (home or away) each team plays in each round. The timetable and the home-away pattern set determine the tournament schedule.

Some round robin scheduling problems consider both the construction of the timetable and of the home-away pattern set. As an example, the traveling tournament problem (TTP) [5] calls for a schedule minimizing the total distance traveled by the teams.

However, either the timetable or the HAP set of the schedule may be fixed and known beforehand in some situations. In the first case, the timetable is given and the problem consists in finding

a feasible HAP set optimizing a certain objective function. The break minimization problem [13] deals with the minimization of the number of breaks (two consecutive home games or two consecutive away games of the same team) in the schedule, while the timetable constrained distance minimization problem [12] calls for the minimization of the total distance traveled.

In the second case, the HAP set is predetermined and a timetable is requested. There is a feasibility issue, since not every HAP set may be associated with a compatible timetable. A necessary condition for feasibility is given in [9]. The problem of constructing a timetable compatible with a given HAP set optimizing certain objective appears as a subproblem in several approaches to solve real-life scheduling problems, see e.g. [10].

A Home-Away Assignment (HAA) [8] is an assignment of a venue to each game. A HAA is balanced if the difference between the number of home games and the number of away games is at most one for every team. If the timetable is fixed, i.e., the round of every game is known, the HAP set and the HAA give the same information and determine the same schedule.

We consider and formulate the problem of scheduling single round robin tournaments with fixed home-away assignments. The venue of each game is known beforehand and the problem consists in determining a timetable minimizing a certain objective function. Variants of this problem find interesting applications in real-life leagues whose DRR tournaments are divided into two SRR phases. Games in the second phase are exactly the same of the first phase, except for the inversion of their venues. Therefore, the venues of the games in the second phase are known beforehand and constrained by those of the games in the first phase. This is the case e.g. of the Chilean soccer professional league [2] and of the German table tennis federation of Lower Saxony [7].

We assume that each team has its own venue at its home city. All teams are initially at their home cities, to where they return after their last away game. The distance $d_{ij} \geq 0$ from the home city of team i to that of team j is known beforehand. A road trip is a sequence of consecutive away games played by a team at the venues of its opponents. This team travels from the venue of one opponent to that of the next, without returning home.

Let G be a HAA, whose elements are ordered pairs of teams. The game between teams i and j is represented either by the ordered pair (i, j) or by the ordered pair (j, i). In the first case, the game between i and j takes place at the venue of team i; otherwise, at that of team j. Consequently, for every two teams i and j, either $(i, j) \in G$ or $(j, i) \in G$.

Given an even number n of teams and a home-away assignment G, the Traveling Tournament Problem with Fixed Venues (TTPFV) consists in finding a compact single round robin schedule compatible with G, such that the total distance traveled by the teams is minimized and no team plays more than three consecutive home games or three consecutive away games. Section 2 describes two integer programming formulations of this problem, with respectively $O(n^3)$ and $O(n^5)$ variables. The strength of their linear relaxations is compared in Section 3. Computational results are presented in Section 4. Concluding remarks are drawn in the last section.

2 Integer programming formulations

2.1 Formulation with $O(n^3)$ variables

We define the following decision variables:

$$z_{tjk} = \begin{cases} 1, & \text{if team } t \text{ plays at home against team } j \text{ in round } k, \\ 0, & \text{otherwise;} \end{cases}$$

$$y_{tij} = \begin{cases} 1, & \text{if team } t \text{ travels from the facility of team } i \text{ to the facility of team } j, \\ 0, & \text{otherwise.} \end{cases}$$

The above variables are used in the formulation (1)-(12) of TTPFV, with $O(n^3)$ variables:

$$\min F_1(z, y) = \sum_{t=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} y_{tij} \tag{1}$$

subject to:

$$\sum_{q=1}^{n-1} z_{tjq} = 1, \qquad \forall\, (t, j) \in G \tag{2}$$

$$\sum_{\substack{j=1 \\ j \neq t}}^{n} (z_{tjk} + z_{jtk}) = 1, \qquad t = 1, \ldots, n, \quad k = 1, \ldots, n-1 \tag{3}$$

$$y_{tij} \geq z_{it,k-1} + z_{jtk} - 1, \qquad t, i, j = 1, \ldots, n \text{ with } t \neq i \neq j, \quad k = 2, \ldots, n-1 \tag{4}$$

$$y_{tit} \geq z_{it,k-1} + \sum_{\substack{j=1 \\ j \neq t}}^{n} z_{tjk} - 1, \qquad t, i = 1, \ldots, n \text{ with } t \neq i, \quad k = 2, \ldots, n-1 \tag{5}$$

$$y_{tti} \geq \sum_{\substack{j=1 \\ j \neq t}}^{n} z_{tj,k-1} + z_{itk} - 1, \qquad t, i = 1, \ldots, n \text{ with } t \neq i, \quad k = 2, \ldots, n-1 \tag{6}$$

$$y_{tti} \geq z_{it1}, \qquad t, i = 1, \ldots, n \text{ with } t \neq i \tag{7}$$

$$y_{tit} \geq z_{it,n-1}, \qquad t, i = 1, \ldots, n \text{ with } t \neq i \tag{8}$$

$$\sum_{q=k}^{k+3} \sum_{\substack{j=1 \\ j \neq t}}^{n} z_{jtk} \leq 3, \qquad t = 1, \ldots, n, \quad k = 1, \ldots, n-4 \tag{9}$$

$$\sum_{q=k}^{k+3} \sum_{\substack{j=1 \\ j \neq t}}^{n} z_{jtk} \geq 1, \qquad t = 1, \ldots, n, \quad k = 1, \ldots, n-4 \tag{10}$$

$$z_{tjk} \in \{0, 1\}, \qquad t, j = 1, \ldots, n, \quad k = 1, \ldots, n-1 \tag{11}$$

$$0 \leq y_{tij} \leq 1, \qquad t, i, j = 1, \ldots, n. \tag{12}$$

The objective function (1) defines the minimization of the total distance traveled by the teams. Constraints (2) ensure that each game occurs exactly once. Constraints (3) guarantee that each team plays one game in each round. Constraints (4) enforce team t to perform a trip from the home city of team i to that of team j if it plays two consecutive away games against teams i and j, in this order. Constraints (5) enforce team t to perform a trip from the home city of team i to its own home city if it has an away game against the latter followed by a home game in the next round. Constraints (6) enforce team t to travel from its own home city to that of team i to play away against the later after a home game in the previous round. Constraints (7) enforce team t to travel to the home city of team i if it plays away against the latter in the first round. Constraints (8) enforce team t to return from the home city of team i if it plays away against the latter in the last round. Constraints (9) establish that team t cannot play more than three consecutive away games, while constraints (10) guarantee that team t cannot play more than three consecutive home games. Constraints (11) define the integrality requirements for the z variables. The y variables act as generalized upper bounds to constraints (4) to (8). Since their costs in the objective function

to be minimized are non-negative, they will always assume the minimal possible value, which is necessarily either 0 or 1. Therefore, the integrality requirements on the y variables may be replaced by lower and upper bounds expressed as constraints (12).

This formulation has $O(n^4)$ constraints: $O(n^2)$ of types (2), (3), (7), (8), (9) and (10), $O(n^3)$ of types (5) and (6), and $O(n^4)$ of type (4).

2.2 Formulation with $O(n^5)$ variables

This formulation considers complete road trips. The variables represent road trips of different sizes, giving a more direct representation of the problem. Three new decision variables are defined and used in the third formulation:

$$w^1_{tik} = \begin{cases} 1, \text{ if team } t \text{ starts, in round } k, \text{ a road trip visiting team } i \text{ and} \\ \quad \text{returning home in round } k+1 \text{ (with } t \neq i) \\ 0, \text{ otherwise;} \end{cases}$$

$$w^2_{tijk} = \begin{cases} 1, \text{ if team } t \text{ starts, in round } k, \text{ a road trip visiting first team } i, \text{ then team } j, \\ \quad \text{and returning home in round } k+2 \text{ (with } t \neq i \neq j) \\ 0, \text{ otherwise;} \end{cases}$$

$$w^3_{tijlk} = \begin{cases} 1, \text{ if team } t \text{ starts, in round } k, \text{ a road trip visiting first team } i, \text{ then team } j, \\ \quad \text{next team } l, \text{ and returning home in round } k+3 \text{ (with } t \neq i \neq j \neq l) \\ 0, \text{ otherwise.} \end{cases}$$

Two dummy rounds (indexed by -1 and 0) are created to simplify the formulation. The variables corresponding to every road trip starting in any of these fictitious rounds is set to 0. The auxiliary costs c_{ij}, c_{ijm}, and c_{ijml} represent the costs of road trips of length one, two, and three performed by team i, respectively:

$$c_{ij} = d_{ij} + d_{ji}, \tag{13}$$

$$c_{ijm} = d_{ij} + d_{jm} + d_{mi}, \tag{14}$$

$$c_{ijml} = d_{ij} + d_{jm} + d_{ml} + d_{li}. \tag{15}$$

The new variables are used to reformulate TTPFV as model (16)-(21) below, with $O(n^5)$ variables. Although the number of variables increases with respect to the previous formulation, we notice that the number of constraints is quite smaller.

$$\min F_3(w^1, w^2, w^3) = \sum_{k=1}^{n-1} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ (j,i)\in G}}^{n} [c_{ij} w^1_{ijk} + \sum_{\substack{m=1 \\ (m,i)\in G}}^{n} (c_{ijm} w^2_{ijmk} + \sum_{\substack{l=1 \\ (l,i)\in G}}^{n} c_{ijml} w^3_{ijmlk})] \tag{16}$$

subject to:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} [\sum_{k\in\{-1,0\}} w^1_{ijk} + \sum_{m=1}^{n} (\sum_{k\in\{-1,0,n-1\}} w^2_{ijmk} + \sum_{l=1}^{n} \sum_{k\in\{-1,0,n-2,n-1\}} w^3_{ijmlk})] = 0 \tag{17}$$

$$\sum_{k=1}^{n-1} \{w^1_{ijk} + \sum_{\substack{m=1 \\ (m,i)\in G}}^{n} [(w^2_{ijmk} + w^2_{imjk}) + \sum_{\substack{l=1 \\ (l,i)\in G}}^{n} (w^3_{ijmlk} + w^3_{imjlk} + w^3_{imljk})]\} = 1, \qquad \forall (j,i) \in G \tag{18}$$

$$\sum_{\substack{j=1\\(j,i)\in G}}^{n}\{w^1_{ijk}+\sum_{\substack{m=1\\(m,i)\in G}}^{n}[(w^2_{ijmk}+w^2_{ijm,k-1})+\sum_{\substack{l=1\\(l,i)\in G}}^{n}(w^3_{ijmlk}+w^3_{ijml,k-1}+w^3_{ijml,k-2})]\}+$$
$$+\sum_{\substack{j=1\\(i,j)\in G}}^{n}\{w^1_{jik}+\sum_{\substack{m=1\\(m,j)\in G}}^{n}[(w^2_{jimk}+w^2_{jmi,k-1})+\sum_{\substack{l=1\\(l,j)\in G}}^{n}(w^3_{jimlk}+w^3_{jmil,k-1}+w^3_{jmli,k-2})]\}=1,$$
$$i=1,\dots,n,\quad k=1,\dots,n-1 \qquad (19)$$

$$\sum_{\substack{j=1\\(j,i)\in G}}^{n}\{w^1_{ijk}+w^1_{ij,k+1}+\sum_{\substack{m=1\\(m,i)\in G}}^{n}[w^2_{ijm,k-1}+w^2_{ijmk}+w^2_{ijm,k+1}+$$
$$+\sum_{\substack{l=1\\(l,i)\in G}}^{n}(w^3_{ijml,k-2}+w^3_{ijml,k-1}+w^3_{ijmlk}+w^3_{ijml,k+1})]\}\le 1,\qquad i=1,\dots,n,\quad k=1,\dots,n-2 \qquad (20)$$

$$\sum_{q=k}^{k+3}\{\sum_{\substack{j=1\\(j,i)\in G}}^{n}[w^1_{ijq}+\sum_{\substack{m=1\\(m,i)\in G}}^{n}((w^2_{ijmq}+w^2_{ijm,q-1})+\sum_{\substack{l=1\\(l,i)\in G}}^{n}(w^3_{ijmlq}+w^3_{ijml,q-1}+w^3_{ijml,q-2}))]\}\ge 1,$$
$$i=1,\dots,n,\quad k=1,\dots,n-4 \qquad (21)$$

$$w^1_{ijk}\in\{0,1\},\qquad i,j=1,\dots,n,\ \text{with } i\neq j,\quad k=-1,\dots,n-1 \qquad (22)$$

$$w^2_{ijmk}\in\{0,1\},\qquad i,j,m=1,\dots,n,\ \text{with } i\neq j\neq m,\quad k=-1,\dots,n-1 \qquad (23)$$

$$w^3_{ijmlk}\in\{0,1\},\qquad i,j,m,l=1,\dots,n,\ \text{with } i\neq j\neq m\neq l,\quad k=-1,\dots,n-1. \qquad (24)$$

The objective function (16) minimizes the total traveled distance. Constraints (17) set to 0 the variables associated to road trips starting at the dummy rounds -1 and 0, to road trips of size two and three starting at the last round and those of size three starting at round $n-2$. Constraints (18) ensure each game occurs exactly once. They represent the fact that each game $(j,i)\in G$ should be played in a road trip of team i formed by one, two or three away games. Constraints (19) enforce that team i is either playing an away game or another team is visiting it in each round. This is achieved by setting to one the sum of all road trips of team i which include round k with the sum of all road trips of other teams which visit i in round k. Constraints (20) forbid team i to be engaged in simultaneous or consecutive (i.e., without returning to its home city) road trips in round k. Constraints (21) state that team i must be outside its home city to play away at least once along every four consecutive rounds. Constraints (22)-(24) guarantee the integrality requirements.

This formulation has $O(n^2)$ constraints of types (18) to (21).

3 Linear relaxation bounds

We refer to the formulations with $O(n^3)$ and $O(n^5)$ variables, respectively, as models N3 and N5. We denote by $\overline{\text{N3}}$, and $\overline{\text{N5}}$, respectively, their linear relaxations. Let $LB3$ and $LB5$ be, respectively, the linear relaxation bounds provided by formulations N3 and N5.

Theorem 1: $LB5 \geq LB3$.
Sketch of the proof: Let $(w^{1^*}, w^{2^*}, w^{3^*})$ be an optimal solution to model $\overline{\text{N5}}$ satisfying constraints (18)-(21). We build a solution $(\hat{y}, \hat{z})$ to model $\overline{\text{N3}}$ by defining:

$$\hat{z}_{tik} = w^{1^*}_{itk} + \sum_{\substack{l=1 \\ (l,i)\in G}}^{n} [(w^{2^*}_{itlk} + w^{2^*}_{ilt,k-1}) + \sum_{\substack{m=1 \\ (m,i)\in G}}^{n} (w^{3^*}_{itlmk} + w^{3^*}_{iltm,k-1} + w^{3^*}_{ilmt,k-2})],$$
$$t, i = 1, \ldots, n, \quad k = 1, \ldots, n-1 \tag{25}$$

$$\hat{y}_{tij} = \sum_{k=1}^{n-1} [w^{2^*}_{tijk} + \sum_{\substack{l=1 \\ (l,t)\in G}}^{n} (w^{3^*}_{tijlk} + w^{3^*}_{tlij,k})], \qquad t, i, j = 1, \ldots, n, \text{ with } t \neq i \neq j \tag{26}$$

$$\hat{y}_{ttj} = \sum_{k=1}^{n-1} \{w^{1^*}_{tjk} + \sum_{\substack{l=1 \\ (l,t)\in G}}^{n} [w^{2^*}_{tjlk} + \sum_{\substack{m=1 \\ (m,t)\in G}}^{n} w^{3^*}_{tjlmk}]\}, \qquad t, j = 1, \ldots, n, \text{ with } t \neq j \tag{27}$$

$$\hat{y}_{tit} = \sum_{k=1}^{n-1} \{w^{1^*}_{tik} + \sum_{\substack{l=1 \\ (l,t)\in G}}^{n} [w^{2^*}_{tlik} + \sum_{\substack{m=1 \\ (m,t)\in G}}^{n} w^{3^*}_{tlmik}]\}, \qquad t, i = 1, \ldots, n, \text{ with } t \neq i. \tag{28}$$

It can be proved by variable substitution that $(\hat{y}, \hat{z})$ satisfies constraints (2)-(12) (i.e., it is feasible to $\overline{\text{N3}}$) and that $F_3(w^{1^*}, w^{2^*}, w^{3^*}) = F_1(\hat{y}, \hat{z})$. The result follows, since $LB5 = F_3(w^{1^*}, w^{2^*}, w^{3^*}) = F_1(\hat{y}, \hat{z}) \geq F_1(y^*, z^*) = LB3$, where (y^*, z^*) is an optimal solution to $\overline{\text{N3}}$. □

4 Computational results

To evaluate and compare the proposed integer programming models, we created 20 home-away assignments for each of the national league (nl) TTP instances in [16] with up to 12 teams. Ten instances in each group have their home-away assignments randomly determined. The other ten instances correspond to balanced home-away assignments: the venues of all games were randomly selected and the iterative method in Knust and Thaden [8] was used to balance the home-away assignments.

The models have been solved by CPLEX 10.0, using the API Concert Technology compiled with g++ 4.0.2. The computational experiments were performed on a Pentium D machine with a 3.0 GHz processor and 2 Gbytes of RAM memory, running under version 2.6.12 of Linux.

The linear relaxations of the two formulations described in Section 2 were solved for all instances. The results are summarized in Table 1. For each value of n, the second and third columns of this table give, respectively, the average lower bounds provided by models N3 and N5 for the feasible instances. The two last columns give the average and maximum gaps between the bounds $LB3$ and $LB5$, i.e., by how much $LB5$ improves upon $LB3$. The average bounds $LB5$ provided by model N5 are far better than those generated by the other formulation. As an example, the average bound $LB5$ for $n = 12$ improves by more than ten times the average bound $LB3$.

We also compared the formulations in terms of the solver capability to find optimal integral solutions. We run CPLEX for at most two hours to solve each instance using each formulation. The results are shown in Table 2. For each number of teams, we give the number of feasible instances, the number of instances solved to optimality using models N3 and N5, the average gap between $LB5$ and the optimal value F_3^*, and the average and maximum times to find an optimal solution

Table 1: Linear relaxation lower bounds

n	avg($LB3$)	avg($LB5$)	avg($(LB5 - LB3)/LB3$) (%)	max($(LB5 - LB3)/LB3$) (%)
4	2064.41	5162.60	150.08	191.59
6	3290.97	13979.0	331.50	396.69
8	4007.89	21477.0	458.58	550.88
10	3762.23	32504.80	764.71	809.42
12	5667.44	58194.74	935.20	1017.92

using model N5. The two models were able to find the optimal solutions to all small instances with $n \leq 6$. For $n = 8$, CPLEX was able to solve only five instances to optimality with model N3, but it was able to find the optimal solutions to all 20 instances with model N5. CPLEX was not able to solve instances with $n > 8$. The quality of the lower bounds $LB5$ (on average at 7.7% from optimality) helped the solver to find optimal solutions in less than one minute of computation time for most instances with $n = 8$.

Table 2: Integral solutions

n	Feasible	opt(N3)	opt(N5)	avg($(F_3^* - LB5)/F_3^*$) (%)	avg. time (s)	max. time (s)
4	20	20	20	0.0	< 0.1	< 0.1
6	17	17	17	2.4	0.2	0.7
8	20	5	20	7.7	25.6	61.8

5 Conclusion

In this paper, we introduced the traveling tournament scheduling problem with fixed venues.Two integer programming formulations for the problem were proposed. The linear relaxation of the formulation with $O(n^5)$ variables provides the strongest bounds and improves the ability of integer programming solvers to find optimal solutions.

The quality of the lower bounds $LB5$ (on average at 7.7% from optimality) helped the solver to find optimal solutions in less than one minute of computation time for all but one instance with $n = 8$. All instances with eight teams were solved in small computation times, but none with 10 teams could be solved in two hours of processing time. This result is similar to those observed for the TTP, which can be easily solved for $n = 6$, but is very hard to solve for $n = 8$.

We are currently investigating the problem of minimizing the number of breaks with fixed venues. Preliminary work leads to conjecture that the minimal number of breaks with balanced fixed home-away assignments is always smaller than or equal to the number of teams in the tournament.

References

[1] A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados (2006), A simulated annealing approach to the traveling tournament problem, *Journal of Scheduling* **9**, 177 – 193.

[2] G. Durán, T.F. Noronha, C.C. Ribeiro, S. Souyris, and A. Weintraub (2006), Branch-and-cut for a real-life highly constrained soccer tournament scheduling problem, In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, 398 – 401.

[3] K. Easton, G.L. Nemhauser, and M. Trick (2002), Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, 319 – 330.

[4] K. Easton, G.L. Nemhauser, and M. Trick (2004), Sports scheduling. In J.T. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chapter 52, 1 – 19, CRC Press.

[5] K. Easton, G.L. Nemhauser, and M.A. Trick (2001), The traveling tournament problem description and benchmarks, In T. Walsh, editor, *Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, 580 – 584, Springer.

[6] M. Henz (1999), Constraint-based round robin tournament planning, In D. De Schreye, editor, *International Conference on Logic Programming*, 545 – 557.

[7] S. Knust (2005), An RCPSP-based model for scheduling a table tennis league, In *Proceedings of the 7th Workshop on Models and Algorithms for Planning and Scheduling Problems*, Siena.

[8] S. Knust and M. von Thaden (2006), Balanced home-away assignments, *Discrete Optimization* **3**, 354 – 365.

[9] R. Miyashiro, H. Iwasaki, and T. Matsui (2002), Characterizing feasible pattern sets with a minimum number of breaks, In E.K. Burke and P. De Causmaecker, editors, *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, 311 – 313.

[10] G.L. Nemhauser and M.A. Trick (1998), Scheduling a major college basketball conference. *Operations Research* **46**, 1 – 8.

[11] R.V. Rasmussen and M.A. Trick (2006), Round robin scheduling - A survey. Technical report, Department of Operations Research, University of Aarhus.

[12] R.V. Rasmussen and M.A. Trick (2006), The timetable constrained distance minimization problem. In J.C. Beck and B.M. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, *Lecture Notes in Computer Science* **3990**, 167 – 181. Springer.

[13] J. Régin (2001), Minimization of the number of breaks in sports scheduling problems using constraint programming, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **57**, 115 – 130.

[14] C.C. Ribeiro and S. Urrutia (2005), An application of integer programming to playoff elimination in football championships, *International Transactions in Operational Research* **12**, 375 – 386.

[15] C.C. Ribeiro and S. Urrutia (2006), Scheduling the brazilian soccer championship, In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, 481 – 3483.

[16] M. Trick (2007), Challenge traveling tournament instances, Online reference at `http://mat.gsia.cmu.edu/TOURN/`, last visited on January 7, 2007.

Unrelated Parallel Machines Scheduling with Resource-Assignable Sequence Dependent Setup Times

Rubén Ruiz
Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Valencia, Spain, rruiz@eio.upv.es

Carlos Andrés
Departamento de Organización de Empresas, Universidad Politécnica de Valencia, Valencia, Spain, candres@omp.upv.es

Setup operations on machines usually need of tooling and/or personnel to be carried out. Therefore, the length of the setup might not only depend on the machine and job sequence, but also on the number of resources assigned. This situation is common in real problems arising in several industries where setup operations in production lines are frequent. These operations are indeed setups whose length can be reduced or extended according to the number of resources assigned to them. We deal with an unrelated parallel machine scheduling problem with a linear combination of total completion time and the total number of resources assigned as an objective. We present a MIP model and some fast dispatching heuristics. We carry out computational experiments to study what characteristics of the problem affect the MIP model performance and to the effectiveness the different heuristics proposed.

Keywords: Machine Scheduling, Production Scheduling, Real World Scheduling, Heuristic Search.

1 Introduction

In this short paper, we schedule n jobs on m unrelated parallel machines. As known, first jobs have to be assigned to machines and second, among the jobs assigned to each machine, a processing sequence must be derived. We denote by p_{ij} the processing time of job j at machine i. We consider machine and sequence dependent setup times which can be separated from processing times. A setup is a non-productive period of time which usually modelizes operations to be carried out on machines after processing a job to leave them ready for processing the next job in the sequence. Therefore, S_{ijk} is the setup time to be carried out on machine i after having processed job j and before processing job k. A clear example of setups stems from the ceramic tile manufacturing sector. At the glazing stage there are several unrelated parallel lines (machines) capable of glazing the different types of ceramic tile lots (jobs). After glazing a lot, the glazing line needs to be cleaned and prepared for the glazing of the next lot or job. This is the setup time which clearly depends on the line type and on the processing sequence.
S_{ijk} is difficult to set in practice as resources are involved. We will denote by R_{ijk} the amount of resources devoted to carrying out setup S_{ijk}. We refer to this as resource-assignable sequence dependent setup times or RASDST in short. Resources are usually costly and a trade-off situation arises between cost and total setup time. Consequently, we are interested in a realistic optimization criterion which minimizes the total production time or flowtime and the resources employed. We denote by T_{Res} the total number of resources assigned to setups. Additionally, given α and β that modelize the importance or cost of each unit of resource and flowtime, respectively, the objective function considered in this paper is $\alpha T_{Res} + \beta \sum_{j=1}^{n} C_j$, where $\alpha, \beta \geq 0$. Such a problem, is obviously $\mathcal{NP}$-Hard since it is a generalization of other simpler $\mathcal{NP}$-Hard problems. For example, the case

where the parallel machines are identical and there are only sequence independent family setup times is already $\mathcal{NP}$-Hard as explained in [10].
In the literature there are several reviews and surveys about scheduling with setup times like the ones of [1] and more recently, [2]. There are also specific surveys dealing with parallel machine settings like those of [4] and [6]. A careful examination of these reviews results in no paper dealing with the problem considered in this work. A related study is due to [3] where a MIP model for a uniform parallel machines problem and earliness/tardiness criteria is proposed. In [12], a mixed integer programming model is proposed for the unrelated machines, sequence dependent setup problem with earliness/tardiness objective. [11] and [5] study similar problems. More recently, [8] have proposed several GRASP-like heuristics for an unrelated machines case with machine and job sequence dependent setup times with makespan criterion.
As we can see, to the best of our knowledge, no scheduling problem has been studied in the literature where the setup times can be resource-assignable.

2 Problem and MIP model formulation

For each possible setup time S_{ijk} we have a set of four values. The first two specify the minimum and maximum quantities of resources that can be assigned to each setup. Therefore, $R^-_{ijk}(R^+_{ijk})$ denote the minimum (maximum) resources assignable to setup S_{ijk}. Similarly, $S^-_{ijk}(S^+_{ijk})$ denote the minimum (maximum) possible setup time. The relation between the actual setup time and the number of resources is assumed to be linear. Therefore, the expression that returns the amount of setup time S_{ijk} as a function of the assigned resources R_{ijk} is the following: $S_{ijk} = S^+_{ijk} - \frac{S^+_{ijk}-S^-_{ijk}}{R^+_{ijk}-R^-_{ijk}}(R_{ijk} - R^-_{ijk}) = S^+_{ijk} + \frac{S^+_{ijk}-S^-_{ijk}}{R^+_{ijk}-R^-_{ijk}}R^-_{ijk} - \frac{S^+_{ijk}-S^-_{ijk}}{R^+_{ijk}-R^-_{ijk}}R_{ijk}$. Using K_1 and K_2 for the first term and for the R_{ijk} multiples, respectively, we have as a result the slope-intercept form of the line relating S_{ijk} and R_{ijk}: $S_{ijk} = K_1 - K_2R_{ijk}$. K_2 is the slope of the line that relates R_{ijk} with S_{ijk}. In other words, K_2 indicates how much the setup time S_{ijk} is reduced by each additional resource. With this we define the following model:

$$\begin{aligned}
X_{ijk} &= \begin{cases} 1, \text{ if job } j \text{ precedes job } k \text{ on machine } i \\ 0, \text{ otherwise} \end{cases} \\
C_{ij} &= \text{Completion time of job } j \text{ at machine } i \\
R_{ijk} &= \text{Resources assigned to the setup between job } j \text{ and job } k \text{ on machine } i
\end{aligned}$$

$$\min \sum_{i\in M}\sum_{j\in N}\sum_{k\in N,k\neq j} \alpha R_{ijk} + \sum_{i\in M}\sum_{j\in N} \beta C_{ij} \tag{1}$$

$$\sum_{i\in M}\sum_{j\in\{0,N\},j\neq k} X_{ijk} = 1, \qquad k \in N \tag{2}$$

$$\sum_{i\in M}\sum_{k\in N,j\neq k} X_{ijk} \leq 1, \qquad j \in N \tag{3}$$

$$\sum_{k\in N} X_{i0k} \leq 1, \qquad i \in M \tag{4}$$

$$\sum_{h\in\{0,N\},h\neq k,h\neq j} X_{ihj} \geq X_{ijk}, \qquad j,k \in N, j \neq k, i \in M \tag{5}$$

$$C_{ik} + V(1 - X_{ijk}) \geq C_{ij} + K_1 - K_2R_{ijk} + p_{ik}, \qquad j \in \{0,N\}, k \in N, j \neq k, i \in M \tag{6}$$

$$R_{ijk} \geq R^-_{ijk}, \qquad j,k \in N, j \neq k, i \in M \tag{7}$$

$$R_{ijk} \leq R^{+}_{ijk}, \qquad j,k \in N, j \neq k, i \in M \tag{8}$$

$$C_{i0} = 0, \qquad i \in M \tag{9}$$

$$C_{ij} \geq 0, \qquad j \in N,\ i \in M \tag{10}$$

$$X_{ijk} \in \{0,1\}, \qquad j \in \{0,N\}, k \in N, j \neq k, i \in M \tag{11}$$

Constraint set (2) ensures that every job has exactly one predecessor. Conversely, constraint set (3) limits the number of successors of every job to one. Set (4) limits the number of successors of the dummy jobs to a maximum of one on each machine, With set (5) we ensure that jobs are properly linked in machines since if a given job j is processed o a given machine i, it must have a predecessor h on the same machine. Constraint set (6) is controls the completion times of the jobs at the machines. Constraint sets (7) and (8) bound the possible assignable resources. Sets (9) and (10) define completion times as 0 for dummy jobs and non-negative for regular jobs, respectively. Finally, set (11) defines the binary variables.

3 Heuristic methods

Shortest Processing Time (SPT) dispatching rules are known to give optimum solutions for simple scheduling problems with the total completion time ($\sum_{j=1}^{n} C_j$) criterion (see [7]). Additionally, all seven heuristics proposed in [11] for the $R/S_{jk}/\frac{1}{n}\sum_{j=1}^{n} w_j C_j$ problem are based on this dispatching rule. It is clear than processing first the shortest jobs results in a lower overall total completion time. Therefore, we also base our proposed heuristics on the SPT rule.

3.1 Shortest Processing Time with Setups resource Assignment (SPTSA)

This procedure returns, for every machine, the jobs assigned to it along with their sequence and the resources assigned to each possible setup.

1. For every job j calculate the machine with the minimum processing time, i.e., $l = \arg\min_{i \in M} p_{ij}$. Store job j, machine l and the minimum processing time p_{lj} in a vector V of size n
2. Sort vector V in increasing order of the minimum processing time of each job on all machines (p_{il})
3. For $j = 1$ to n do
   (a) Take from $V[j]$ the job k and the machine l to which assign the job k
   (b) Assign resources to the setup between the last job assigned to machine l and job k
   (c) Assign job k to machine l

A final issue remains about the quantity of resources to assign to each setup (step 3b). For the moment we assign the average resources $(R^{+}_{ijk} + R^{-}_{ijk})/2$.

3.2 Shortest Processing and Setup Time with Setups resource Assignment (SPSTSA)

SPSTSA is very similar to SPTSA, the only difference is that average setups are also considered along with the minimum processing times for deriving the dispatching order of jobs. Therefore, the first two steps of the SPTSA are modified as follows:

1. For every job j calculate the the minimum processing time and average setup time for all machines, i.e., calculate an index I as follows:

$$I_j = \min_{i \in M} \left(p_{ij} + \frac{1}{n-1} \sum_{k=1, k \neq j}^{n} \frac{S^+_{ijk} + S^-_{ijk}}{2} \right) \tag{12}$$

   Let l be the machine where the minimum I_j has been obtained. Store job j, machine l and I_j in a vector V of size n

2. Sort vector V in increasing order of I_j

3.3 Dynamic Job Assignment with Setups resource Assignment (DJASA)

Both previous dispatching rules have a main drawback. In an unrelated parallel machine problem, the minimum processing time, or minimum index I for two jobs can be on the same machine l. Therefore, assigning both jobs to l might not be the best solution. DJASA solves this problem by dynamically considering all pending jobs and all machines at each iteration. The procedure is the following:

1. Add all jobs to a list of unscheduled jobs ρ
2. While $\rho \neq \varnothing$ do
   (a) For every pending job in ρ ($\rho_{(j)}$) and for every machine $i \in M$ do
      i. Temporarily assign resources to the setup between job $\rho_{(j)}$ and the previous job assigned to machine i
      ii. Assign $\rho_{(j)}$ to machine i
      iii. Calculate the objective $\alpha T_{Res} + \beta \sum_{j=1}^{n} C_j$ after the resource and job assignment
   (b) Let j and l be the job and machine that has resulted in the minimum overall objective increase, respectively
   (c) Assign resources to the setup between job j and the previous job assigned to machine l
   (d) Assign job j to l
   (e) Remove job j from ρ

As with SPTSA and SPSTSA, average resources are assigned. For SPTSA and SPSTSA, the overall complexity is $\mathcal{O}(n \log n)$. In DJASA, a total of $\frac{n(n+1)}{2} m$ steps are needed, which results in a $\mathcal{O}(n^2 m)$ complexity.

3.4 Optimal resource assignment

Once the assignment of the different jobs to the unrelated parallel machines and the sequence among them is known we can do a better re-assignment of resources. A high slope K_2 means that large reductions in setup time are obtained by assigning additional resources. Given the linearity of the relation between S_{ijk} and R_{ijk}, the same reductions are obtained for each additional resource in the range $[R^-_{ijk}; R^+_{ijk}]$. As a result, the following assignment procedure can be applied:

1. For each machine i and for each job k assigned to i except the last do
   (a) Let j be the predecessor of k at i or 0 if job k is the first job assigned to i

(b) Let h be the number of successors of job k assigned to machine i

(c) The resources R_{ijk} assigned to the setup between jobs j and k at machine i are re-assigned according to the following expression:

$$R_{ijk} = \begin{cases} R^+_{ijk}, \text{ if } K_2 \cdot h \cdot \beta > \alpha \\ R^-_{ijk}, \text{ otherwise} \end{cases} \quad (13)$$

It is straightforward to see that the previous re-assignment of setup resources is optimal. Reducing the setup time one unit will reduce the completion times of all jobs processed after the setup in one unit as well. In a simple case where $K_2 = \alpha = \beta = 1$, if two jobs are assigned to a given machine after a setup, increasing the resources of this setup in one unit will improve the objective. In this case, and giving the linearity of K_2, the maximum resources (minimum setup) should be assigned to obtain the maximum savings. This optimal resource assignment has a complexity of only $\mathcal{O}(n)$ and and be easily applied to the resulting solution of the previous methods.

4 Computational Evaluation

A comprehensive benchmark has been defined for the evaluation. We generate all the data: processing times p_{ij} and the minimum and maximum resources and setups (R^-_{ijk}, R^+_{ijk}, S^-_{ijk} and S^+_{ijk}). The MIP model is not expected to be usable for large instances. Therefore, two sets of instances are defined. For the small instances, all the following combinations of n and m are tested: $n = \{6, 8, 10\}$ and $m = \{3, 4, 5\}$. For the large instances the combinations are $n = \{50, 75, 100\}$ and $m = \{10, 15, 20\}$. In all cases, the processing times are generated according to a uniform distribution in the range $[1, 99]$ as it is usual in the scheduling literature. Two resources settings are considered where the resources are uniformly distributed in the ranges $U[1, 3; 3, 5]$ and $U[1, 5; 5, 10]$, respectively. For the setup times we also work with two combinations $U[1, 50; 50, 100]$ and $U[50, 100; 100, 150]$. A total of 720 instances are generated. We set the weights for the objective function as $\alpha = 50$ and $\beta = 1$. The rationale behind this decision is that resources are usually scarce and more expensive than each unit of flowtime.

We construct a model in LP format for each instance in the small set. The resulting 360 models are solved with CPLEX 9.1 on a Pentium IV 3.2 GHz computer with 1 Gbyte of RAM memory. Commercial solvers have been used for parallel machine problems with setups by [3] and [12] and for more complex problems by [9]. We solve all small instances with two different time limits, 5 and 60 minutes. As we will see, not all instances are solved within these time limits. Consequently, for each run we record a categorical variable which we refer to as "type of solution" with two possible values 0 and 1. Type 0 means that an optimum solution was found and type 1 means that the time limit was reached and at the end a feasible integer solution was found. Due to the large data set, and to obtain sound conclusions, It is important to carry out statistical testing. We have carried out statistical testing with decision trees. This approach was used recently by [9] also for the analysis of a MIP model. The resulting tree is shown in Figure 1. Some direct conclusions can be drawn from the decision tree. For example, all instances with $n = 6$ are solved to optimality under both stopping time criteria. However, for $n = 8$, only a few optimum solutions are obtained within 5 minutes CPU time. For 60 minutes CPU time, all $n = 8$ instances are solved. For $n = 10$ no optimum solution could be found for any instance even with 60 minutes CPU time. The effect of other factors is also easy to see. Increasing the number of machines m results in instances that are easier to solve. The effect of the resources is also interesting. Having more resources (i.e., $U[1, 5; 5, 10]$)

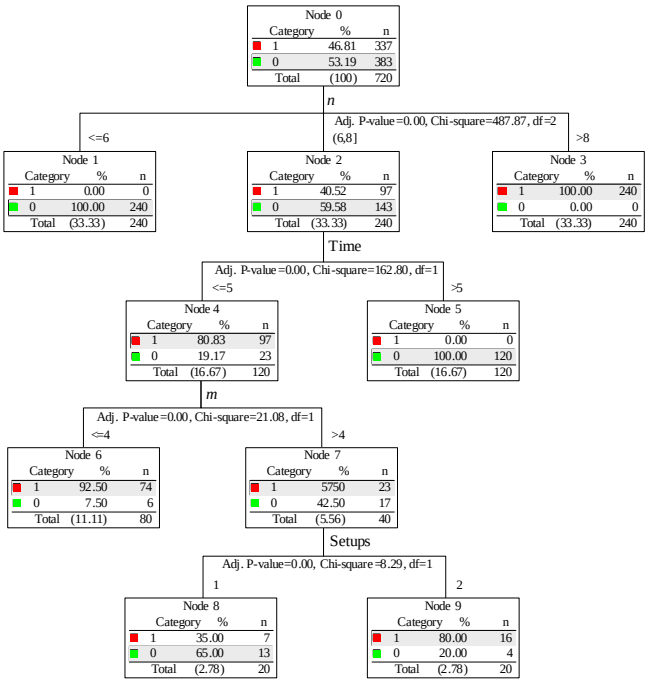


Figure 1: Decision tree of the MIP model results. Response variable type of outcome. Setups 1 and 2 refer to $U[1,50;50,100]$ and $U[50,100;100,150]$, respectively.

results in instances that are easier to solve. On the other hand, larger setups ($U[50,100;100,150]$) result in harder instances. All these observations are important since they allow us to characterize which elements from the novel unrelated parallel machines RASDST problem affect performance. Overall, the efficiency of the MIP model is good, specially when compared to results from the literature. [3] solved problems of up to $n = 10$ and $m = 4$ but on a simpler setting with uniform parallel machines. Their MIP model contains significantly less variables due to the triangular law of inequality assumption in the setup times, i.e., [3] assume that $S_{ijk}+S_{ikl} \geq S_{ijl}$. Obviously, in our RASDST problem this assumption does not hold. [12] work with an unrelated parallel machines case and use a similar variable definition to the one employed in this paper. In this case, the largest size of problem solved is $n = 9$ and $m = 3$ needing almost 5,400 seconds of CPU time in this latter case. Considering that in our model, apart from the job assignment and the job sequencing we have also the assignable-resources, we conclude that the MIP model performance is good.

We proceed now to test the performance of the proposed heurisics. Recall that there are three main methods where we assign average resources: SPTSA, SPSTSA and DJASA. We can apply the optimal resource assignment procedure to these methods which results in three more heuristics: SPTSA*, SPSTSA* and DJASA*. As a result, there are six different methods. All heuristics have been coded in Delphi 2006 and have been tested on the same computer used for testing the MIP model.
We test each method on all instances, where we measure the average percentage increase ($AVRPD$) over the optimum solution (or best solution found) given by the MIP model in the small instances set. For the large instances we test the average percentage deviation over the best solution found by all heuristics. All instances, as well as the best known solutions are available from

http://www.upv.es/gio/rruiz. Table 1 gives the $AVRPD$ values for the small and large instances. For reasons of space, only global average results are reported. As expected, DJASA is the

	SPTSA	SPSTSA	DJASA	SPTSA*	SPSTSA*	DJASA*
Average small	**90.6**	**88.6**	**36.5**	**63.1**	**61.9**	**16.7**
Average large	**53.1**	**53.2**	**22.2**	**32.3**	**32.5**	**6.6**

Table 1: Average percentage deviations from MIP model solutions for the six tested heuristics. Small and large instances.

best method over SPSTSA and this later one is in turn better than SPTSA. The $AVRPD$ values drop strongly for DJASA in all situations. As expected, the optimal resource assignment procedure improves the results always since the procedure cannot deteriorate the solutions. Although not shown, higher number of resources and lower duration of setups results in higher deviations. This is an interesting result since just contradicts the findings of the MIP model. As with the case of the MIP model, the previous table contains observed averages alone. We carried out full experimental analysis using the ANOVA technique. In the experiment we control n, m, R, S and Algorithm as factors. From the results of the experiments, all controlled factors as well as many interactions of order two are significant. We find that most of the observed differences of Table 1 are statistically significant.
A much desirable characteristic of the proposed heuristics is their speed. Among all results gathered with the six algorithms in the large instances, the largest observed elapsed CPU time (not wall time) has been 15.6 milliseconds. For all instances in the small set, the CPU time needed is below of what can be reliably measured. For the large instances, the slowest method is DJASA* but as said, the CPU times are in the low milliseconds range. Recall that the proposed heuristics have a very low complexity. These heuristics can therefore be used in real-time scheduling environments.

5 Conclusions

In this paper, a novel complex scheduling problem is considered. The setting consists in a number of unrelated parallel machines where machine and sequence dependent separable setup times are present. The novelty resides in that the duration of the setups depend on assignable resources. This characteristic has been named as Resource Assignable Sequence Dependent Setup Times or RASDST in short. A combination of total assigned resources and total completion time is used as a criterion. A MIP model, three dispatching heuristics and an optimal resource assignment rule have been proposed. The results indicate that the dynamic job dispatching rule with the optimal resource assignment procedure, a method referred to as DJASA*, is the best heuristic among those proposed. Furthermore, the CPU times of the proposed heuristics are negligible and are therefore useful in real-time scheduling scenarios. We consider this is a good starting work on a practical new type of problems where setup times are resource-assignable and the future research lines are numerous and promising.

References

[1] A. Allahverdi, J.N.D. Gupta and T. Aldowaisan (1999), A review of scheduling research involving setup considerations, *OMEGA, The international Journal of Management Science*, **27**(2), 219 – 239.

[2] A. Allahverdi, C.T. Ng, T.C.E. Cheng and M.Y. Kovalyov (2007), A survey of scheduling problems with setup times or costs, *European Journal of Operational Research*, In press.

[3] N. Balakrishnan, J.J. Kanet and S.V. Sridharan (1999), Early/tardy scheduling with sequence dependent setups on uniform parallel machines, *Computers & Operations Research* **26**(2), 127 – 141.

[4] T.C.E. Cheng and C.C.S. Sin (1990), A state-of-the-art review of parallel machine scheduling research, *European Journal of Operational Research* **47**(3), 271 – 292.

[5] D.W. Kim, K.H. Kim, W. Jang and F.F. Chen (2002), Unrelated parallel machine scheduling with setup times using simulated annealing, *Robotics & Computer Integrated Manufacturing* **18**(3-4), 223 – 231.

[6] E. Mokotoff (2001), Parallel machine scheduling problems: a survey, *Asia-Pacific Journal of Operational Research* **18**(2),193 – 242.

[7] M. Pinedo (2002), *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Upper Saddle, N.J, second edition.

[8] G. Rabadi, R.J. Moraga, and A. Al-Salem (2006), Heuristics for the unrelated parallel machine scheduling problem with setup times, *Journal of Intelligent Manufacturing* **17**(1), 85 – 97.

[9] R. Ruiz, F. Sivrikaya Şerifoğlu and T. Urlings (2007), Modeling realistic hybrid flexible flowshop scheduling problems, *Computers & Operations Research*, In press.

[10] S.T. Webster (1997), The complexity of scheduling job families about a common due date, *Operations Research Letters* **20**(2), 65 – 74.

[11] M.X. Weng, J. Lu and H. Ren (2001), Unrelated parallel machines scheduling with setup consideration and a total weighted completion time objective, *International Journal of Production Economics* **70**(3), 215 – 226.

[12] Z. Zhu and R. Heady (2000), Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach, *Computers & Industrial Engineering* **38**(2), 297 – 305.

Minimizing the Total Weighted Number of Late Jobs with Late Deliveries in Two-Level Supply Chains

George Steiner, Rui Zhang
McMaster University, Hamilton, Ontario, Canada, {steiner, zhangr}@mcmaster.ca

Supply chain scheduling is an emerging area of research. We study the upstream supplier's batch scheduling problem in a supply chain defined by Hall and Potts [1]. The supplier has to manufacture multiple products and deliver them to customers in batches. There is an associated delivery cost with each batch. The objective of the supplier is to minimize the sum of the weighted number of late jobs and batch delivery costs. Citing technical difficulties in scheduling late jobs for delivery, the authors in [1] gave a pseudopolynomial solution for the version of the problem where only early jobs get delivered. We present a dynamic programming algorithm with pseudopolynomial complexity that also schedules the late jobs for delivery. This proves that the problem is $\mathcal{NP}$-hard only in the ordinary sense.
Keywords: [Batch Scheduling, Delivery Scheduling, Machine Scheduling, Theoretical Scheduling].

1 Introduction

Supply chain management has been one of the most important topics in manufacturing research over the last fifteen years. Most of the supply chain literature focuses on inventory control issues on the strategic level, using stochastic models, however, Thomas and Griffin [2] point out in their review of the literature that for many products logistics expenditures can constitute as much as 30% of their cost. This underlines the need for research dealing with supply chain problems on the operational level, using deterministic rather than stochastic models. The recently emerging research area of supply chain scheduling tries to address this problem.

As it is a new area for research, there are relatively few papers dealing specifically with scheduling problems in supply chains. Hall and Potts presented the first paper [1] on this important topic. They study a variety of scheduling, batching and delivery problems in supply chains with the objective of minimizing the overall scheduling and delivery cost. Chen and Hall [3] extend these to supply chains with assembly-type manufacturing systems. Some of the issues studied in these papers are related to previous work on coordinating production and distribution systems. We mention here the papers by Williams [4], and Lee and Chen [5], which consider the integration of transportation time and capacity issues with scheduling decisions. Selvarajah and Steiner [6, 7] develop exact and approximation algorithms for the supplier's problem of minimizing the sum of the total weighted flow time and batch delivery costs. Dawande *et al.* [8] study conflict and cooperation issues in supply chain scheduling. Agnetis *et al.* [9] look at the problem of rescheduling to resolve conflicts between the supplier's and the manufacturers' ideal schedules.

One model introduced by Hall and Potts [1] is the supplier's problem of minimizing the sum of the weighted number of late jobs (late job penalties) and batch delivery costs. They show that the problem is $\mathcal{NP}$-hard in the ordinary sense and present a pseudopolynomial dynamic programming algorithm for its solution. For the case of identical weights, the algorithm becomes polynomial. Citing technical difficulties in scheduling late jobs for delivery [1, 10], they gave a pseudopolynomial solution for the version of the problem where only early jobs get delivered. In this paper, we present a different dynamic programming algorithm with pseudopolynomial complexity that also schedules

the late jobs for delivery. This proves that the problem with late deliveries is also $\mathcal{NP}$-hard only in the ordinary sense.

2 Problem definition and preliminaries

We start by defining our problem in detail. We have a two-level supply chain between the supplier and the set of customers, $M = \{1, 2, \cdots, m\}$. The supplier has to make and deliver the jobs $J_i = \{(1, i), (2, i), \cdots, (n_i, i)\}$ for customer $i \in M$. The whole job set is $J = \bigcup_{i \in M} J_i$ and $|J| = n = \sum_{i \in M} n_i$. Jobs are processed and delivered in batches to each customer. A batch can contain jobs only for the same customer and has a sequence-independent batch-setup time s_i and batch-delivery cost q_i for customer $i \in M$. Job $(k, i) \in J_i$ has processing time $p_{(k,i)}$ and due date $d_{(k,i)}$ with $p_{(k,i)} \leq d_{(k,i)}$. If it is delivered after $d_{(k,i)}$, the job incurs a late penalty of $w_{(k,i)}$ (the weight). Our objective is to find the schedule which processes and delivers *all* jobs and minimizes the sum of the weighted number of late jobs and delivery costs.

The problem is related to the classic scheduling problem of minimizing the total weighted number of late jobs that has been extensively studied in the last forty years. Moore's algorithm [11] solves the unweighted problem in $O(nlogn)$ time for n jobs. The weighted version of the problem, however, is contained in Karp's original list of $\mathcal{NP}$-hard problems [12]. Sahni [13] presents a dynamic programming algorithm and a fully polynomial time approximation scheme (FPTAS) to maximize the weighted number of early jobs. Gens and Levner [14] develop a FPTAS for the original minimization version of the weighted problem, which requires $O(n^3/\varepsilon)$ time to find an $\varepsilon-$approximate solution. Later [15] they improve the time complexity to $O(n^2 logn + n^2/\varepsilon)$. Hochbaum and Landy [16] study the batching version of the weighted problem, without delivery costs, and present a pseudopolynomial algorithm for it. For the same problem, Brucker and Kovalyov [17] present another pseudopolynomial algorithm that can be converted into a FPTAS with time complexity $O(n^3 logn + n^3/\varepsilon)$. This batching problem is similar to our problem with $m = 1$ customer, since both assume that late jobs also get produced, but the the presence of delivery costs in our problem represents considerable complication even when $m = 1$.

The original model in [1] did not include setup times for the batches. We decided to include these because this makes the model more realistic and this way our problem also represents a proper and substantial generalization of the model studied in [16]. The fact that late jobs also have to be produced and delivered makes the structure of potential optimal schedules more complex. Since once a job is scheduled to be late we incur the same lateness penalty no matter when it gets delivered, we could decide to deliver all late jobs for a customer at the same time in a *designated late batch* at the end of the schedule. Such a schedule would automatically incur the delivery cost for this all-late batch. On the other hand, it may be possible to deliver all the late jobs for a customer together with some early jobs and save the delivery cost for the last all-late batch. We call a batch *early* if it contains at least one job that is delivered on time. A batch is *pure early* if it contains only early (on-time) jobs and is *mixed early* if it also contains some late jobs. The following easy-to-prove propositions describe structural properties of optimal schedules. Several of them are adaptations of results known for batching schedules without deliveries.

Proposition 2.1. *There exists an optimal schedule such that jobs in the same batch are processed without any interruption by any other jobs.*

Proof. It is clear that there is no benefit from interrupting the processing of jobs in a batch by jobs from another batch. □

Proposition 2.2. *There exists an optimal schedule in which early jobs are ordered in EDD (earliest due date) order within any batch.*

Proof. The completion time of the last job processed in a batch is the completion (and delivery) time for all jobs in this batch. This completion time cannot exceed the smallest due date among all early jobs in this batch. Thus any order, and therefore the EDD order, produces the same set of early jobs within this batch. □

Proposition 2.3. *There exists an optimal schedule in which all early batches are ordered in EDD order with respect to the smallest due date of early jobs in each batch.*

Proof. A simple interchange argument between batches proves this proposition. □

Proposition 2.4. *There exists an optimal schedule in which all early jobs for the same customer are ordered in EDD order over all early batches for this customer.*

Proof. Suppose that there are two early batches for the same customer, batch i and batch j with completion times (defined as the completion time of the last job in the batch) $t_i < t_j$. If there is an early job in batch i with a larger due date than the due date of another early job in batch j, then this job can be moved from batch i to batch j without increasing the cost of the schedule. After we moved all such jobs into the later batches, we can apply Proposition 2.2 to obtain the desired schedule. □

Proposition 2.5. *There exists an optimal schedule in which* all *late jobs (if any) for customer i $(i = 1, ..., m)$ are delivered either in the last early batch for customer i or in the designated late batch for this customer.*

Proof. If the optimal schedule has some late jobs in the designated late batch for the customer, then we can move all other late jobs for this customer into the designated batch without increasing the cost of the schedule. Moving a late job from a (mixed) early batch to the designated late batch can only reduce the completion time of any other job (for any customer), which was scheduled before the designated late batch. If the designated late batch is empty for customer i in the optimal schedule, then moving all late jobs for this customer into his last early batch does not increase the cost and can only reduce the completion time of any other job. □

Proposition 2.5 implies that only the last early batch can be mixed early for a customer. Thus, when scheduling jobs for a customer i, we can restrict our search to two types of schedules:

1. Type I in which *all* late jobs are scheduled in the designated late batch for customer i (and all early batches are pure early);

2. Type II in which all late jobs are scheduled at the end of the *last* early batch for customer i, thereby saving the delivery cost for the designated late batch, which is empty. This also implies that job (n_i, i) is always scheduled in the last early batch for customer i in a Type II schedule.

3 The dynamic programming algorithm

Assume that jobs in each set J_i are in EDD order, and they will be scheduled by our algorithm in this order. The state space used to represent a partial schedule in our dynamic programming algorithm DP is described by six entries $\{\mathcal{J}^k, \mathcal{T}, t, h, d, v\}$ defined as follows:

$\mathcal{J}^k$: the m dimensional vector $\mathcal{J}^k = (k_1, \cdots, k_m)$ records that the total number of jobs scheduled in this partial schedule is $k = \sum_{i \in M} k_i$, where k_i is the number of jobs already scheduled for customer i ($i \in M$);

$\mathcal{T}$: the m dimensional vector $\mathcal{T} = (\tau_1, \tau_2, \cdots, \tau_m)$, where τ_i is the total length of jobs scheduled in the designated late batch for customer $i \in M$ in this partial schedule;

t: the completion time of the last job processed in the last early batch in this partial schedule;

h: the last batch of this partial schedule belongs to customer $h \in M$, and this batch will be referred to as the *current batch*;

d: the due date of the last early batch in this partial schedule, defined as the smallest due date of early jobs in this batch;

v: the objective value for this partial schedule calculated recursively.

The dynamic programming algorithm follows the framework of forward recursive state generation suggested by Sahni [13] and starts from the empty state where no job has been scheduled yet, i.e., $k = 0$. The set $\mathcal{S}^{(k)}$ contains all generated partial schedules that contain k jobs, $k = 1, 2, \cdots, n$. First, Type I partial schedules are generated in non-decreasing order of their cardinality k. All these schedules are implicitly assumed to have a designated late batch for each customer i at the end of the schedule, that is not empty when $\tau_i > 0$. From Proposition 2.5, we know that when we want to schedule all late jobs for customer i in an early batch (Type II schedule), we only need to consider the *last* early batch for customer i, and this batch must contain job (n_i, i). Thus we consider *rescheduling* all late jobs for customer i into an early batch only *when* we are scheduling the job (n_i, i).

Algorithm DP

[*Initialization*] Set $(\mathcal{J}^0, \mathcal{T}, 0, 0, 0, 0) \in \mathcal{S}^{(0)}$, where $k_i = 0 \in \mathcal{J}^0$, $\tau_i = 0 \in \mathcal{T}$ $\forall i \in M$, and set $\mathcal{S}^{(k)} \leftarrow \emptyset$, for $k = 1, 2, \cdots, n$.

[*Generation*] Generate set $\mathcal{S}^{(k)}$ for $k = 1$ to $k = n$ from $\mathcal{S}^{(k-1)}$ by the following procedures:

For each $(\mathcal{J}^{k-1}, \mathcal{T}, t, h, d, v) \in \mathcal{S}^{(k-1)}$, where $k - 1 = \sum_{i \in M} k_i$ and $\mathcal{J}^{k-1} = (k_1, k_2, \cdots, k_m)$

For each $i \in M$ with $k_i < n_i$ /* *There are still some unscheduled jobs for customer i.*

Set $k_i = k_i + 1$ and generate $\mathcal{J}^k = (k_1, \cdots, k_i, \cdots, k_m)$;

1. If $t + p_{(k_i,i)} \leq d$ and $i = h$, set $t^{'} = t + p_{(k_i,i)}$ and $\mathcal{S}^{(k)} \leftarrow \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{T}, t^{'}, h, d, v)$ /* We can schedule job (k_i, i) as early in the current batch;
2. If $t + p_{(k_i,i)} + s_i \leq d_{(k_i,i)}$, set $t^{'} = t + p_{(k_i,i)} + s_i$, $h^{'} = i$, $d^{'} = d_{(k_i,i)}$, $v^{'} = v + q_i$ and $\mathcal{S}^{(k)} \leftarrow \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{T}, t^{'}, h^{'}, d^{'}, v^{'})$ /* Schedule job (k_i, i) as an early job starting a new batch for customer i;
3. If $\tau_i > 0$, set $\tau_i^{'} = \tau_i + p_{(k_i,i)}$ and $\tau_j^{'} = \tau_j$ for $j \neq i$ to define $\mathcal{T}^{'}$, $v^{'} = v + w_{(k_i,i)}$ and $\mathcal{S}^{(k)} \leftarrow \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{T}^{'}, t, h, d, v^{'})$ /* Schedule job (k_i, i) as a late job in the nonempty designated late batch for customer i;'
4. If $\tau_i = 0$, set $\tau_i^{'} = p_{(k_i,i)}$ and $\tau_j^{'} = \tau_j$ for $j \neq i$ to define $\mathcal{T}^{'}$, $v^{'} = v + w_{(k_i,i)} + q_i$ and $\mathcal{S}^{(k)} \leftarrow \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{T}^{'}, t, h, d, v^{'})$ /* Schedule job (k_i, i) late by putting it into the previously empty designated late batch for customer i;

5. If $k_i = n_i,\ i = h,\ \tau_i > 0$, and $t + p_{(k_i,i)} + \tau_i \leq d$, then set $t^{'} = t + p_{(k_i,i)} + \tau_i$, $\tau_i^{'} = 0$ and $\tau_j^{'} = \tau_j$ for $j \neq i$ to define $\mathcal{T}^{'}$, $v^{'} = v - q_i$ and $\mathcal{S}^{(k)} \leftarrow \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{T}^{'}, t^{'}, h, d, v^{'})$ /* Reschedule job (k_i, i) and all late jobs from the designated late batch for customer i into the current batch;
6. If $k_i = n_i$, $\tau_i > 0$, and $t+p_{(k_i,i)}+\tau_i+s_i \leq d_{(k_i,i)}$, then set $t^{'} = t+p_{(k_i,i)}+\tau_i+s_i$, $\tau_i^{'} = 0$ and $\tau_j^{'} = \tau_j$ for $j \neq i$ to define $\mathcal{T}^{'}$, $d^{'} = d_{(k_i,i)}$ and $\mathcal{S}^{(k)} \leftarrow \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{T}^{'}, t^{'}, h, d^{'}, v)$ /* Reschedule job (k_i, i) and all late jobs from the designated late batch for customer i into a newly started early batch for customer i;

End For

End For

[*Selection*] Select the $(\mathcal{J}^n, \mathcal{T}, t, h, d, v)$ with smallest v as our optimal result from set $\mathcal{S}^{(n)}$ and trace back to get the optimal schedule.

Consider the partial schedules generated in steps 1-4: Note that the early jobs are always scheduled in EDD order within a batch. It is easy to see that if we temporarily ignored (deleted) the jobs that got scheduled into the designated late batch for each customer in these schedules, then we would obtain *all* partial schedules in which only the early jobs are delivered. Thus our algorithm implicitly generates all partial schedules that would need to be considered for solving the version of the problem in which only early jobs get delivered, i.e., the problem solved in [1].

In steps 5 and 6, where we reschedule all late jobs for customer i together with job (n_i, i) into the last early batch for customer i, we update $v^{'} = v - q_i$ and $v^{'} = v$, respectively. This will account correctly for the delivery component of our cost, since we are reducing the number of batches for customer i by 1 in the first case, and it does not change in the second case (we empty the designated late batch, but we set up a new early batch). When we do this rescheduling, however, some of the rescheduled late jobs from the designated late batch may become early again. It means that the resulting $v^{'}$ value may be greater than the true objective value for this partial schedule, since the objective value would need to be reduced by the tardiness penalty of any previously late job that becomes early. To do this adjustment correctly, we would also need to store for each partial schedule in its state the total weight of late jobs in its designated late batch for each customer, which would add an additional factor of W^m to the size of our state space, where $W = \sum_{(k,i)\in J} w_{(k,i)}$ is the total weight. We show below, however, that this can be avoided.

Consider a partial schedule σ and state $(\mathcal{J}^k, \mathcal{T}, t, h, d, v)$ generated in step 5 or 6 of the algorithm. Suppose job (j, i) is first scheduled as a late job in its designated late batch, but ends up being early when we reschedule job (j, i) into the last early batch for customer i in σ. The situation is demonstrated in the two schematic Gantt charts of Figure 1. The first chart shows the partial schedule before we schedule job (n_i, i) as early in step 5 or 6. The batch boundaries are indicated by longer and heavier lines. The second chart depicts the schedule σ generated in step 5 by scheduling job (n_i, i) as early at the end of the current batch followed by rescheduling all late jobs from the designated late batch by adding them immediately after (n_i, i) into the current batch. (The schedule generated in step 6 would look similar, except job (n_i, i) would be the first job starting a new current batch.).

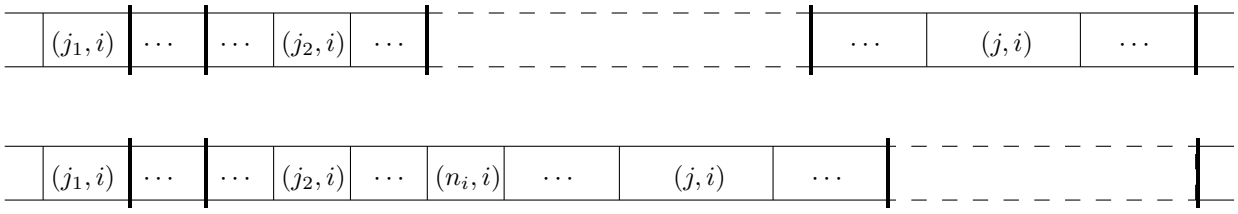


Figure 1

Since job (j,i) became early after rescheduling, we have $\widetilde{t} \leq d_{(j,i)} \leq d_{(n_i,i)}$, where $\widetilde{t}$ is the completion time of the last batch for customer i. Let (j_2,i) be the first originally scheduled early job in the batch whose due date is at least $d_{(j,i)}$. Move job (j,i) to the left in the batch by inserting it immediately before (j_2,i), and repeat this forward insertion operation for every job (j,i) that became early in σ. The Gantt chart of the resulting schedule is depicted in Figure 2. Job (j_1,i) denotes the last job in the preceding batch for customer i (if any).

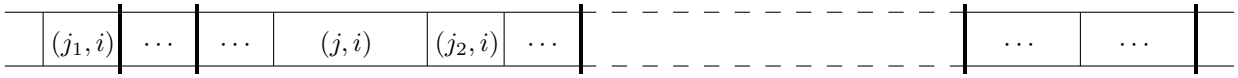


Figure 2: $d_{(j_1,i)} \leq d_{(j,i)}$

If $d_{(j_1,i)} > d_{(j,i)}$ for one of the moved jobs (j,i), then move (j_1,i) to immediately follow each such (j,i) in the last batch. The resulting schedule is shown in Figure 3. If there are any other jobs in the preceding batches for customer i whose due date is greater than such a $d_{(j,i)}$, repeat this operation for these. If we again consider only the early jobs in the resulting schedule, they are in EDD order after these operations. Thus the resulting early job set with this partitioning must have been generated some time during the execution of steps 1-4 in our algorithm, since the algorithm considers every early schedule.

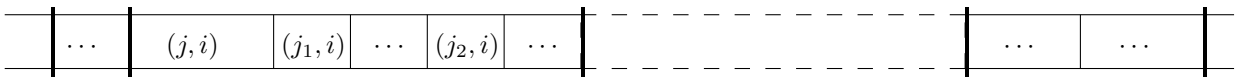


Figure 3: $d_{(j_1,i)} > d_{(j,i)}$

After performing the above transformations for every job (j,i) that may have become early in schedule σ, we obtain a schedule σ' whose early job set is exactly the same as the early job set of the schedule σ produced by the rescheduling step 5 or 6. Moreover, the early jobs are in EDD order in this transformed schedule σ' after the transformations, and its early job set with the same partitioning must have been generated and considered during the execution of steps 1-4 in our algorithm. Furthermore, notice that job (n_i,i) is contained in this early job set, since it was early in σ, and the set of late jobs in the schedule σ' (implicitly scheduled in its designated late batch) would be the same as the set of late jobs in σ. Therefore, right after the algorithm generated σ' (in one of steps 1-4), it would have considered rescheduling all the late jobs for customer i from its designated late batch by putting them right after (n_i,i) in σ'. Let us call this last schedule with the late jobs attached right after (n_i,i) schedule σ''. Notice that every state variable for σ'', except the objective value v'', must have the same value as the corresponding state variable for σ. Furthermore, the v'' value contains the lateness penalties exactly for the late jobs in σ'' (and σ) and thus correctly represents the objective value for the corresponding state and $v > v''$. Thus we have proved the following lemma.

Lemma 3.1. *Suppose that during the execution of rescheduling step 5 or 6 in Algorithm DP, a state* $(\mathcal{J}^k, \mathcal{T}, t, h, d, v)$ *is generated for partial schedule* σ *in which some previously late jobs become early and thus* v *does not correctly represent the total lateness cost of the corresponding schedule* σ*. Then the algorithm must also generate a state* $(\mathcal{J}^k, \mathcal{T}, t, h, d, v^{''})$ *and schedule* $\sigma^{''}$ *such that* $v^{''}$ *correctly represents the cost of* $\sigma^{''}$*, the schedules* σ *and* $\sigma^{''}$ *have the same early and late job sets and* $v^{''} < v$*.*

Lemma 3.1 means that even if our algorithm generates a partial schedule σ with corresponding state $(\mathcal{J}^k, \mathcal{T}, t, h, d, v)$ such that v does not represent the correct lateness cost of the schedule, this will create no problem because in this case, it will always generate an alternative schedule $\sigma^{''}$ with lower and correct cost. Since in algorithm DP we choose the state with smallest v from $\mathcal{S}^{(n)}$ as the final solution, the algorithm will correctly identify an optimal schedule at the end.

In order to study the complexity of the above algorithm, we introduce some further notation. Let $L = \sum_{(k,i)\in J} p_{(k,i)}$ be the total processing time of all jobs, $Q = \sum_{i\in M} n_i q_i$ an upper bound on the total delivery cost for all schedules, $S = \sum_{i\in M} n_i s_i$ an upper bound on the total batch-setup time for all schedules, and $D = \max_{(k,i)\in J}\{d_{(k,i)}\}$ the maximum due date over all jobs.

Theorem 3.1. *Algorithm DP finds an optimal solution in* $O(n^{m+1}L^m[min\{D, L+S\}][W+Q])$ *time and space.*

Proof. Let us calculate the total number of possible different states. Each state is represented by six entries: $\{\mathcal{J}^k, \mathcal{T}, t, h, d, v\}$, $k = 0, 1, \cdots, n$. We know that $\mathcal{J}^k = (k_1, k_2, \cdots, k_m)$ with $k = \sum_{i=1}^{i=m} k_i$. Any $k_i \in \mathcal{J}^k$ can take at most $k+1$ different values, $\{0, 1, \cdots, k\}$, and the first $m-1$ k_i values and k also determine k_m. Therefore, we have at most $(k+1)^{m-1}$ different vectors $\mathcal{J}^k$. Because the maximum k value is n, the upper bound for the number of $\mathcal{J}^k$ is $(n+1)^{m-1}$ for all k. We also have $\mathcal{T} = (\tau_1, \tau_2, \cdots, \tau_m)$, and any $\tau_i \in \mathcal{T}$ can take at most $p_i + 1$ different values from 0 to $p_i = \sum_{(k,i)\in J_i} p_{(k,i)}$. Therefore, we can have at most $\prod_{i=1}^{i=m}(p_i+1)$ different vectors $\mathcal{T}$. Because L is an upper bound of p_i for all i, the total number of different $\mathcal{T}$ vectors is bounded by $(L+1)^m$. As both h and d carry information for the job that is scheduled as the first job in the current early batch, we have at most n different combinations. Since t is the total processing and setup time of the early batches, its value must be between 0 to $\min\{D, L+S\}$. Since v is the objective value, it is upperbounded by $[W+Q]$. In summary, at the beginning of the kth iteration there are at most $(n+1)^{m-1}(L+1)^m n[\min\{D, L+S\}][W+Q]$ different states that need to be considered. For each state, there are at most m candidate jobs to be considered as the next job. For each job we generate at most six different new states in Steps 1-6. Therefore, the time complexity of each iteration is $O(m(n+1)^{m-1}(L+1)^m n[\min\{D, L+S\}][W+Q])$, which is $O(n^m L^m[\min\{D, L+S\}][W+Q])$ for fixed m. Since we have n iterations, the overall running time is $O(n^{m+1}L^m[\min\{D, L+S\}][W+Q])$. □

Remark 3.1. We note that the algorithm could also be used for solving a version of the problem in which the delivery of a batch for customer i requires a delivery time $del_i > 0$. We simply would need to reduce the due date for each job (j, i) by del_i before running the algorithm.

4 Conclusions and further research

We presented a pseudopolynomial dynamic programming algorithm (for fixed number of customers) to minimize the sum of the total weighted number of late jobs and batch delivery costs in a two-level supply chain when late jobs also have to be produced and delivered. This proves that the problem is $\mathcal{NP}$-hard in the ordinary sense and gives rise to the question whether there is also an FPTAS for the problem. Another open question is whether the dynamic programming algorithm can also be extended for the problem in multi-level supply chains.

Acknowledgement

The authors thank Dr.N. Hall for useful discussions on the subject. This research was supported in part by NSERC grant No. OG1798-03.

References

[1] N. G. Hall and C. N. Potts (2003), Supply chain scheduling: Batching and delivery, *Operations Research* **51**(4), 566 – 584.

[2] D. J. Thomas and P.M. Griffin (1996), Coordinated supply chain management, *European Journal of Operations Research* **94**, 1 – 15.

[3] Z.L. Chen and N.G. Hall (2000), Supply chain scheduling: Assembly systems, *Working Paper.*

[4] J.F. Williams (1981), A hybrid algorithm for simultaneous scheduling of production and distribution in multi-echelon structures, *Management Science* **29**, 77 – 92.

[5] C.Y. Lee and Z.L. Chen (2001), Machine scheduling with transportation considerations, *Journal of Scheduling* **4**, 3 – 24.

[6] E. Selvarajah and G. Steiner (2006), Batch scheduling in a two-level supply chain - a focus on the supplier, *European Journal of Operational Research* **173**(1), 226 – 240.

[7] E. Selvarajah and G. Steiner (2006), Batch scheduling in customer-centric supply chains, *Journal of the Operations Research Society of Japan* **49**(3), 174 – 187.

[8] M. Dawande, H. N. Geismar, N. G. Hall, and C. Sriskandarajah (2006), Supply chain scheduling: Distribution systems, *Production and Operations Management* **15**(2), 243 – 261.

[9] A. Agnetis, N. G. Hall, and D. Pacciarelli (2006), Supply chain scheduling: Sequence coordination, *Discrete Applied Mathematics* **154**(15), 2044 – 2063.

[10] N. G. Hall (2006), Private communication.

[11] J.M. Moore (1968), An n job, one machine sequencing algorithm for minimzing the number of late jobs, *Management Science* **15**, 102 – 109.

[12] R.M. Karp (1972), Reducibility among combinatorial problem, 85–103, In R.E. Miller and Thatcher J.W., editors, *Complexity of Computer Computations*, Plenum Press, New York.

[13] S. K. Sahni (1976), Algorithms for scheduling independent tasks, *Journal of the ACM*, **23**(1), 116 – 127.

[14] G. V. Gens and E. V. Levner (1979), Discrete optimization problems and efficient approximate algorithms, *Engineering Cybernetics* **17**(6), 1 – 11.

[15] G. V. Gens and E. V. Levner (1981), Fast approximation algorithm for job sequencing with deadlines, *Discrete Applied Mathematics* **3**(4), 313 – 318.

[16] D. S. Hochbaum and D. Landy (1994), Scheduling with batching: minimizing the weighted number of tardy jobs, *Operations Research Letters* **16**, 79 – 86.

[17] P. Brucker and M.Y. Kovalyov (1996), Single machine batch scheduling to minimize the weighted number of late jobs, *Mathematical Methods of Operations Research* **43**, 1 – 8.

Genetic Algorithm for Late Work Minimization in a Flow Shop System

Malgorzata Sterna, Jacek Blazewicz
Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland,
{malgorzata.sterna, jacek.blazewicz}@cs.put.poznan.pl
Erwin Pesch
Institute of Information Systems, FB 5 - Faculty of Economics, University of Siegen, Hoelderlinstrasse 3,
57068 Siegen, Germany, erwin.pesch@uni-siegen.de

The work concerns a genetic algorithm for the flow shop scheduling problem with release times and the late work criterion, which is NP-hard. The proposed meta-heuristic method minimizes the number of units of job processing times executed after their given due dates. We describe the components of this approach and present the analysis of test results. Computational experiments, preceded by an extensive tuning process, were performed for different classes of problem instances in terms of the distribution of release times and due dates over time.

Keywords: Shop Scheduling, Late Work Criterion, Meta-heuristic Search, Genetic Algorithm.

1. Introduction

The late work objective function [1], minimizing the total number of tardy units of particular activities executed in the system, takes into account mostly the amount of late work not the quantity of the delay. It combines the idea of two classical performance measures [2, 9]: the total tardiness and the number of late jobs. Similarly as tardiness, late work increases with a job delay, but when the job becomes totally late, the penalty remains on the certain level (determined by the job processing time) as for the number of late jobs parameter. The concept of late work is strictly related to the imprecise computation model (e.g. [20]). It can be applied for all cases when jobs or activities can be delayed, but the size of their delayed parts influences the quality of a schedule.

First, the criteria based on late work (cf. [26] for a survey) were investigated for parallel identical and uniform [1, 3] and for single machine(s) environments [14–16, 18, 19, 22, 23]. Then, these performance measures were applied for the dedicated machines cases. The studies on the shop systems concentrated mainly on the two-machine weighted cases with a common due date and they resulted in proving the ordinary NP-hardness of open [4], flow [5] and job [8] shop problems. Results of computational experiments are available only for the two-machine flow shop case for which list scheduling approach [6] and three meta-heuristics (tabu search, simulated annealing and variable neighborhood search) were proposed [7]. The non-weighted late work criterion was investigated only for the open-shop problem [4, 25] and, recently, for the flow shop case [21, 28]. There is also an example of a more practical application of the total late work performance measure to the flexible manufacturing environment, modeled as the extended job shop system [27].

In the presented work, we propose a genetic algorithm [12, 17, 24] for the flow shop scheduling problem with the arbitrary number of machines, job release times and the total late work criterion, which is known to be NP-hard [21]. Evolutionary algorithms have not been applied for the late work minimization so far.

In Section 2, the problem under consideration is defined formally. Section 3 describes particular components of the proposed genetic algorithm. Section 4 presents results of computational experiments. The work is concluded in Section 5.

2. Problem formulation

The flow shop problem (cf. e.g. [2, 9]) investigated in the work, $F \mid r_j \mid Y$, concerns executing a set of n jobs $J = \{J_1, J_2, ..., J_n\}$ on a set of m dedicated machines $M = \{M_1, M_2, ..., M_m\}$. Each job $J_j \in J$

consists of m tasks T_{ij} (i = 1, ..., m), which have to be processed without preemption for p_{ij} time units by machine M_i. Each job J_j has to be first executed by machine M_1, then, M_2, M_3, ... M_m and its execution can start at earliest at release time r_j. Moreover, each job can be processed on at most one machine at the same time and each machine can perform at most one job at the same time. Since we consider the permutation flow shop model, the jobs execution order on all machines is identical, and a schedule can be described as a single sequence of jobs from set J.

The goal is to minimize the total late work in the system. The late work Y_j for job J_j is determined as the sum of late parts of tasks T_{1j}, ..., T_{mj}, executed after due date d_j. Denoting the completion time of task T_{ij} as C_{ij}, the late work for job J_j is given by:

$$Y_j = \sum_{i=1}^{m} \min\{\max\{0, C_{ij} - d_j\}, p_{ij}\}.$$

Consequently, the total late work for the entire flow shop system is determined as:

$$Y = \sum_{j=1}^{n} Y_j .$$

The problem stated above is NP-hard, due to NP-hardness of the two-machine flow shop case $F2 \mid d_j = d \mid Y$, which has been proven [21] by a transformation from the partition problem. Due to the intractability of problem $F \mid r_j \mid Y$, it can be solved efficiently only by approximation approaches.

3. Genetic algorithm

The genetic algorithm (GA) proposed for the problem under consideration is based on the general framework for this meta-heuristic [12, 17, 24]. The method starts with an initial population of chromosomes, where each chromosome represents a single solution to the problem. In each iteration, a current population is modified by applying two operators: crossover and mutation, then a new population is selected for the next iteration of the algorithm. The method stops after reaching a termination condition.

3.1. Solution representation

A population analyzed in each iteration of GA is a set of chromosomes, which indirectly represent solutions to problem $F \mid r_j \mid Y$. A chromosome is transformed to a schedule by applying the list scheduling method (cf. e.g. [2]).

The list scheduling algorithm (LA) is a constructive procedure, which builds a feasible solution iteratively extending a partial schedule and selecting jobs according to a given priority dispatching rule (PDR, cf. e.g. [13]). In each iteration, the method determines the set of available jobs (i.e. jobs whose release times have been exceeded) and it assigns the job with the highest priority (with regard to PDR) to the machines, as soon as they are ready for executing another job. Consequently, the list scheduling algorithm performs n steps to construct a feasible solution for a set of n jobs.

In the presented approach, the priority dispatching rule used in LA is not fixed. In each iteration a different rule can be applied. It is determined by a chromosome, which is a sequence of $(n-1)$ priority dispatching rules (R_1, R_2, ..., R_{n-1}) [11]. A schedule corresponding to a particular chromosome is constructed by the list scheduling algorithm by assigning to the machines a job selected from a set of available jobs according to priority dispatching rule R_i at iteration i (the last selection is obvious).

As it was mentioned, chromosomes consist of static or dynamic priority dispatching rules. Among static priority dispatching rules, based only on the problem's parameters, we apply the following ones:

- LPT – the longest job processing time $\sum_{i=1}^{m} p_{ij}$,
- SPT – the shortest job processing time $\sum_{i=1}^{m} p_{ij}$,
- LPTP – the longest processing time of the first task in a job p_{1j},

- SPTP – the shortest processing time of the first task in a job p_{1j},
- EDD – the earliest job due date d_j,
- SA – the smallest allowance $(d_j - r_j)$,
- SA/PT – the smallest allowance per job processing time $(d_j - r_j) / \sum_{i=1}^{m} p_{ij}$.

Assuming that t_1 denotes the partial schedule length on machine M_1 and c_{ij} denotes the completion time of task T_{ij}, if job J_j would be assigned to the machines as the next one, we use the following dynamic priority dispatching rules:

- ML – the minimum predicted lateness $(d_j - (t_1 + \sum_{i=1}^{m} p_{ij}))$,
- MS – the minimum slack $(d_j - t_1) / \sum_{i=1}^{m} p_{ij}$,
- MnCT – the minimum job completion time c_{mj},
- MxCT – the maximum job completion time c_{mj},
- STR – the smallest time remaining $(d_j - c_{mj})$.

In each iteration, the genetic algorithm analyzes a population of p chromosomes. The initial population contains twelve uniform chromosomes, i.e. chromosomes which are built from only one priority dispatching rule, and $(p - 12)$ chromosomes generated randomly.

3.2. Genetic operators

The implemented genetic algorithm uses a two-parent crossover operator, which constructs two new chromosomes from two parent ones. We propose two operators: one-point and two-point crossover. In the first case (C1), prefixes of a randomly chosen length are interchanged between two chromosomes. In the latter one (C2), subsequences between two randomly chosen positions are taken into account.

The mutation operator modifies a single chromosome by changing the sequence of priority dispatching rules randomly. We propose two mutation operators. The first one (M1) interchanges PDRs between two randomly chosen positions. The latter one (M2) influences the structure of a chromosome significantly – it randomly selects one PDR in a chromosome and replaces all its instances with another randomly selected PDR.

The number of chromosomes selected from a population for the recombination (as parents for crossover operator) is determined by crossover rate (the percentage of the best chromosomes which form pairs for crossover). Similarly, the number of chromosomes being an input for the second genetic operator – mutation – is determined by mutation rate (the probability of being selected for mutation).

3.3. Search process

In each iteration of the genetic algorithm, a population of chromosomes is extended with new chromosomes obtained by the recombination. The quality of all individuals in the population is determined by the criterion value, i.e. the total late work for a schedule corresponding to a certain chromosome. We apply GA with the constant size of the population and the steady state evolution process with the ranking approach. At the end of each iteration, we select the constant number of the best chromosomes from a current population to be analyzed in the following steps of the method.

The search process is continued until the termination condition is reached. We use two stopping criteria: the number of generated populations (i.e. the number of iterations) and the number of populations without improvement in the solution quality.

4. Computational experiments

The genetic algorithm for problem $F \mid r_j \mid Y$ was implemented in Java 1.5 and tested on PC (Intel® Pentium® M740 1.73GHz 1GB RAM). The computational experiments were divided into two

phases. First, the method was tuned to determine appropriate values of its control parameters, then the behavior of the tuned GA was analyzed for different classes of input data.

4.1. Input data

The proposed genetic algorithm was tested for the classical benchmarks for the flow shop scheduling problem [29], which were completed with additional parameters, i.e. job release times and due dates (following the idea from [10]).

Release times r_j were taken from uniform distribution over [0, αLB], where LB denotes the lower bound of the schedule length, known for particular benchmarks [29], and α is a control parameter. Due dates d_j were determined with regard to job release times as:

$$d_j = r_j + \sum_{i=1}^{m} p_{ij} + U[N - NR/2, N + NR/2],$$

where $N = (1–T)$LB, T and R are control parameters, U denotes a uniform distribution. Thus, due date d_j for a certain job was equal to its earliest feasible completion time ($r_j + \sum_{i=1}^{m} p_{ij}$) increased with a certain factor.

Depending on the parameters values (α, R, T), it is possible to generate instances with tight and loose r_j and d_j values. The tight values of r_j (TR) correspond to the situation when all release times are taken from a small range of values around time zero, while the loose values of r_j (LR) model the situation when release times are spread in time. Similarly, tight values of d_j (TD) correspond to the case when due dates are very close to the smallest feasible completion time of jobs. On the contrary, the loose values of d_j (LD) model the situation when due dates are not so restrictive.

Based on 36 selected classical benchmarks [29] of 12 different sizes (in terms of the numbers of jobs and machines, $n \in [20, 500]$ and $m \in [5, 20]$), four groups of instances were generated differing in due dates and release times of jobs:

- TR-TD with $\alpha = 0.1$, $R = 1$, $T = 0.8$,
- LR-TD with $\alpha = 0.4$, $R = 1$, $T = 0.8$,
- TR-LD with $\alpha = 0.1$, $R = 0.5$, $T = 0.5$,
- LR-LD with $\alpha = 0.4$, $R = 0.5$, $T = 0.5$.

Consequently, the computational experiments were performed for 144 instances of various characteristics.

4.2. Tuning process

The tuning process was devoted to determining the values of control parameters for the genetic algorithm, which result in its highest efficiency. Obviously, the efficiency of a meta-heuristic depends on input instances and the optimal setting of control values cannot be determined. However, the tuning process makes it possible to adjust a meta-heuristic to input data.

Tuning was performed with all available instances for which the average improvement of the criterion value with regard to its initial value was calculated. We tested 32 configurations with different crossover operators (C1 or C2), mutation operators (M1 or M2), crossover rate (5% or 10%), mutation rate (5% or 10%) and the size of the population (100 or n). The search was stopped after 100 iterations.

Computational experiments showed that depending on the control parameters values, the average criterion improvement varied from 7.89% to 17.77% of its initial value. Similarly the average running time varied from 1801 to 16273 ms. This diversity of results confirms the usefulness of the tuning process. After the careful analysis of its results, during the main experiments, we used two-point crossover operator (C2), mutation operator introducing deep changes into a chromosome (M2), higher crossover and mutation rates (10%) and the size of population equal to 100 individuals. In additional experiments, we compared two termination conditions, i.e. the number of iterations without improvement equal to 100 and to n. For the further experiments, we chose

the first value, which ensured the same quality of obtained solutions within significantly shorter running time than the second one.

4.3. Results of computational experiments

The computational experiments confirmed the strong influence of the problem data on the efficiency of the genetic algorithm (Tables 1 and 2). The large standard deviation values resulted from including into particular classes of instances, instances of various sizes.

Table 1. The differences in the criterion value for different classes of input instances

Class of instances	Difference between initial and final solutions [%]		Difference between initial solution and best uniform chromosome [%]	
	Average value	Standard deviation	Average value	Standard deviation
TR-TD	5.45	3.75	0.95	1.71
LR-TD	6.44	4.44	1.09	3.08
TR-LD	18.71	7.08	2.17	4.45
LR-LD	45.05	25.27	3.26	8.32
all tests	18.91	20.82	1.87	5.07

The genetic algorithm improved the quality of initial solutions by nearly 19% on average, but its efficiency strictly depended on the type of problem instance.

Class TR contains instances in which most jobs become available at the same time – there are small differences in the release times. Thus, the set of available jobs is numerous and, additionally, most jobs have a similar priority, especially with regard to the PDRs involving the job release times (this effect is additionally strengthened by tight due dates). In consequence, the genetic algorithm was less efficient in improving the solution's quality than for the other classes of instances. On the contrary, class LR includes instances for which release times are spread in time – and there are fewer jobs available at the same time in comparison to LR and the priorities of jobs are more diverse (especially for loose due dates). Since jobs have various priorities for particular priority dispatching rules, modifying the sequence of these rules in chromosomes allowed GA to achieve larger improvements of the criterion value.

However, the influence of due dates on the method's efficiency seems to be more important than release times (which is understandable, since the criterion value is calculated with regard to d_j). Class TD contains instances with tight due dates, in which there is a very small allowance window for particular jobs. In consequence, most of jobs are late, independently of the sequence of their execution, and GA could only slightly improve the criterion value. In the case of loose due dates the allowance windows are much wider and the criterion value is strongly influenced by the sequence of jobs. Thus it was possible to obtain higher improvement in the value of the performance measure by changing the execution order of jobs.

Summing up, the largest criterion value improvement (about 45%) was obtained for instances with loose release times and due dates (LR-LD), but, simultaneously, the efficiency of GA was the least stable, which is expressed in the large standard deviation value.

The similar observations can be formulated with regard to the quality of uniform chromosomes, i.e. list solutions for particular priority dispatching rules. The quality of uniform chromosomes was slightly worse (of 1.87%) than the quality of random individuals in the initial population (Table 1). Surprisingly applying the fixed strategy of assigning jobs to machines was less efficient than a random selection. The less restrictive release times and due dates are, the difference is more visible (the difference increases from 0.95% to 3.26%). For the instances with the tight due dates and release times (TR-TD), the structure of a chromosome was less important – the quality of all solutions in the initial population was similar. The test results showed that uniform chromosomes were worse than random ones, so list scheduling solutions for fixed PDRs were worse than initial ones and, consequently, they were worse than final schedules. This means that the genetic algorithm constructed much better solutions than the list scheduling heuristic (by more than 20% of the criterion value on average).

Table 2. The time efficiency of GA for different classes of input instances

Class of instances	Number of iterations		Running time [ms]	
	Average	Standard deviation	Average	Standard deviation
TR-TD	371	344	9510	21068
LR-TD	591	1042	20590	81221
TR-LD	728	831	27509	61012
LR-LD	424	416	21432	60788
all tests	529	725	19760	59843

The instance type influenced not only the quality of a schedule constructed by GA, but also its time efficiency (cf. Table 2). The running times of GA obviously reflect the number of populations generated by the method. For class TR-LD, GA performed the smallest number of iterations, since it was not able to improve the initial solution significantly. For LR-LD, the criterion improvement was very large, but it was achieved quite fast, then the algorithm felt into a local optimum and terminated. For the remaining two classes of instances, GA explored more solutions, especially for instances with loose due dates (TR-LD), performing more iterations. Similarly as in the case of the quality of solutions, the large standard deviation values for running times resulted from analyzing instances of various sizes within particular classes. The efficiency of the genetic algorithm depends not only on the characteristic of data set, but it is also significantly influenced by the instance size.

Fig. 1. The criterion value improvement with regard to the initial solution for different sizes of input instances

The improvement of the criterion value with regard to an initial solution decreased with the size of instances, from about 30% to about 11% (Fig. 1). The larger instances were analyzed, the smaller part of the solution space was explored by the genetic algorithm, which influenced the quality of the final solution. Moreover, for large sets of jobs modifications of schedules, performed within a single iteration, caused smaller changes in the criterion value, than it was observed for sets of jobs with a lower cardinality.

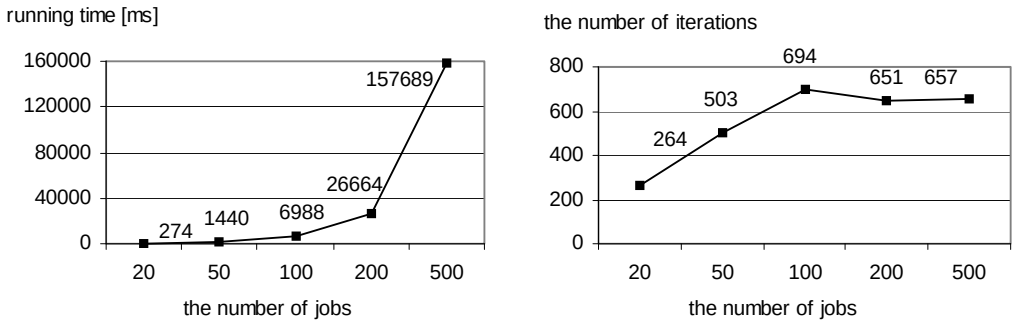


Fig. 2. The time efficiency of GA for different sizes of input instances

The running time of GA increased significantly with the size of instances, but the number of iterations performed by the method (i.e. generated populations) became stabilized for large prob-

lems (Fig. 2). The behavior of GA for large instances was dominated by the list scheduling method (cf. Table 3). It has to be applied for every chromosome in a population and it requires $O(n^2)$ time. For this reason for large problem instances, the list scheduling algorithm consumed the majority of computational time. For smaller instances, the recombination was the second most time-consuming operation. Applying the mutation operator required incomparably shorter running time than using the crossover operator. In this latter case, the genetic operator was applied to a subset of the best chromosomes, which required to rank all individuals in the population, and moreover, after recombination, duplicates were detected and removed from a current population.

Table 3. The usage of running time for particular components of the genetic algorithm

The number of jobs	The usage of running time for particular GA's components [%]			
	Crossover	Mutation	List scheduling algorithm	Other components
20	45.88	1.11	47.83	5.18
50	23.27	0.44	74.55	1.74
100	10.9	0.18	88.25	0.67
200	4.16	0.07	95.54	0.23
500	1.26	0.02	98.65	0.06

5. Conclusions

In the paper we presented the genetic algorithm for the flow shop problem with release times and the total late work criterion. Computational experiments performed for different classes of input instances allowed us to formulate interesting conclusions on the efficiency of the method under consideration.

The proposed genetic algorithm represents a solution to the problem as a sequence of priority dispatching rules used by the list scheduling algorithm during constructing a schedule. Within the future research, we want to perform additional computational experiments in order to investigate the influence of particular priority dispatching rules on the solution quality. Moreover, the method will be improved and completed with additional components.

Acknowledgements. We thank Bartosz Stefaniak for his effort in implementing and testing the approach investigated within the reported research. The first author was partially supported by KBN grant.

References

[1] J. Blazewicz (1984), Scheduling preemptible tasks on parallel processors with information loss, *Technique et Science Informatiques* **3(6)**, 415 – 420.

[2] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz (2001), *Scheduling computer and manufacturing processes*, 2 ed., Springer, Berlin-Heidelberg-New York.

[3] J. Blazewicz, G. Finke (1987), Minimizing mean weighted execution time loss on identical and uniform processors, *Information Processing Letters* **24**, 259 – 263.

[4] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2004), Open shop scheduling problems with late work criteria, *Discrete Applied Mathematics* **134**, 1 – 24.

[5] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2005), The two-machine flow-shop problem with weighted late work criterion and common due date, *European Journal of Operational Research* **165**(2), 408 – 415.

[6] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2005), A comparison of solution procedures for two-machine flow shop scheduling with late work criterion, *Computers and Industrial Engineering* **49**, 611 – 624.

[7] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2007), Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date, *Computers and Operations Research*, doi:10.1016/j.cor.2006.03.021, in press.

[8] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2007), A note on two-machine job shop with weighted late work criterion, *Journal of Scheduling* **10**(2), 87 – 95.
[9] P. Brucker (1998), *Scheduling algorithms*, 2 ed., Springer, Berlin-Heidelberg-New York.
[10] E. Demirkol, S. Mehta, R. Uzsoy (1998), Benchmarks for shop problems, *European Journal of Operational Research* **109**, 137 – 141.
[11] U. Dorndorf, E. Pesch (1995), Evolution based learning in a job shop scheduling environment, *Computers and Operations Research* **22**, 25 – 40.
[12] D.E. Goldberg (1989) *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading.
[13] R. Haupt (1989), A survey of priority rule-based scheduling, *OR Spektrum* **11**, 3–16.
[14] A.M.A. Hariri, C.N. Potts, L.N. Van Wassenhove (1995), Single machine scheduling to minimize total late work, *INFORMS Journal on Computing* **7**, 232 – 242.
[15] D.S. Hochbaum, R. Shamir (1990), Minimizing the number of tardy job unit under release time constraints, *Discrete Applied Mathematics* **28**, 45 – 57.
[16] D.S. Hochbaum, R. Shamir (1991), Strongly polynomial algorithms for the high multiplicity scheduling problem, *Operations Research* **39**(4), 648 – 653.
[17] J.H. Holland (1975), *Adaptation in natural and artificial systems*, University of Michigan, Ann Harbor.
[18] R.B. Kethley, B. Alidaee (2002), Single machine scheduling to minimize total late work: a comparison of scheduling rules and search algorithms, *Computers and Industrial Engineering* **43**, 509 – 528.
[19] M.Y. Kovalyov, C.N. Potts, L.N. Van Wassenhove (1994), A fully polynomial approximation scheme for scheduling a single machine to minimize total weighted late work, *Mathematics of Operations Research* **19**(1), 86 – 93.
[20] J.Y.-T. Leung (2004), Minimizing total weighted error for imprecise computation tasks and related problems, Chapter 34 in: J.Y.-T. Leung (ed.), *Handbook of scheduling*: *algorithms*, *models*, *and performance analysis*, CRC Press, Boca Raton.
[21] B.M.T. Lin, F.C. Lin, R.C.T. Lee (2006), Two-machine flow-shop scheduling to minimize total late work, *Engineering Optimization* **38**(4), 501 – 509.
[22] C.N. Potts, L.N. Van Wassenhove (1991), Single machine scheduling to minimize total late work, *Operations Research* **40**(3), 586 – 595.
[23] C.N. Potts, L.N. Van Wassenhove (1991), Approximation algorithms for scheduling a single machine to minimize total late work, *Operations Research Letters* **11**, 261 – 266.
[24] K. Sastry, D. Goldberg, G. Kendall (2005), Genetic algorithms, Chapter 4 in: E.K. Burke, G. Kendall (eds.), *Search methodologies: introductory tutorials in optimization and decision support methodologies*, Springer, New York.
[25] M. Sterna (2000), *Problems and algorithms in non-classical shop scheduling*, Ph.D. Thesis, Scientific Publishers of the Polish Academy of Sciences, Poznan.
[26] M. Sterna (2006), *Late work scheduling in shop systems*, Dissertations 405, Publishing House of Poznan University of Technology, Poznan.
[27] M. Sterna (2007), Late work minimization in a small flexible manufacturing system, *Computers and Industrial Engineering* **52**(2), 210 – 228.
[28] M. Sterna (2007), Dominance relations for two-machine flow-shop problem with late work criterion, *Bulletin of the Polish Academy of Sciences* **55**(1), 59 – 69.
[29] E. Taillard (1993), Benchmarks for basic scheduling problems, *European Journal of Operational Research* **64**, 278 – 285.
(http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement. dir/ordonnancement.html)

Cyclic Scheduling of Tasks with Unit Processing Time on Dedicated Sets of Parallel Identical Processors

Pŗemysl Šůcha[1] and Zdeněk Hanzálek[1,2]
[1]Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic, {suchap, hanzalek}@fel.cvut.cz
[2]Merica, Czech Republic, hanzalek@merica.cz

This paper presents an integer linear programming (ILP) model for cyclic scheduling of tasks with unit processing time. Our work is motivated by digital signal processing (DSP) applications on FPGA (Field-Programmable Gate Array) architectures hosting several kinds of identical arithmetic units. These hardware resources can be formalized as dedicated sets of parallel identical processors. We propose a method to find an optimal periodic schedule of DSP algorithms on such architectures. The accent is put on the efficiency of the ILP model. We show advantages of the model in comparison with common ILP model on benchmarks and randomly generated instances.

Keywords: Multi-processor Scheduling, Cyclic Scheduling, Digital Signal Processing Applications.

1 Introduction

Integer Linear Programming (ILP) formulations find their application in scheduling as well as other areas in engineering. The challenge for researchers is to provide faster solutions for ILP based approaches. The modeling problem plays a crucial role on the solution efficiency.

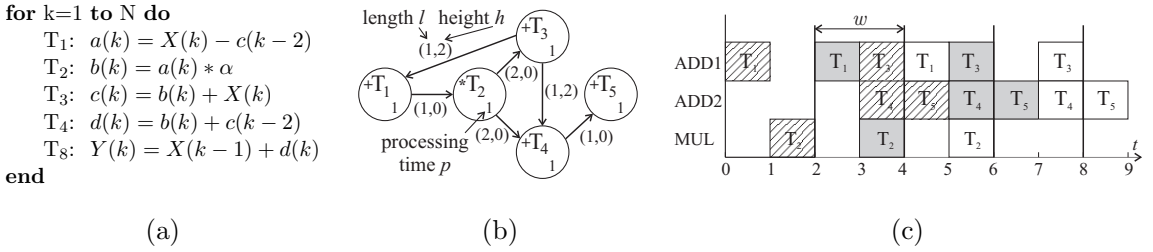
In this paper, we concentrate on the efficiency of the ILP model used for automatic parallelization of computation loops. Our work is motivated by DSP (Digital Signal Processing) applications executed on FPGAs (Field Programmable Gate Arrays) hosting several kinds of identical arithmetic units. These hardware resources can be formalized as dedicated processors.

The DSP application is usually implemented in the form of an iterative loop. A parallel implementation of the loop requires each operation of the loop to be mapped on the arithmetic unit at a given time, which is formulated as a cyclic scheduling problem. The arithmetic units operating with real numbers are usually pipelined, i.e. the input data are usually passed to the unit in one clock cycle, while the result of a computation is available after several clock cycles denoted as the *input–output latency* of the unit formalized as *precedence delays*.

An example of the arithmetic library for FPGA is FP32 [1] implementing a 32-bit floating point number system and compliant with IEEE standards. Another library using a logarithmic number system arithmetic is HSLA [2]. In both cases, rather complex arithmetic is required. Therefore, scheduling of such dedicated HW resources has to carefully consider the algorithm structure, in order to achieve the desired performance of the application. Scheduling also helps to choose the appropriate arithmetic library prior to the algorithm implementation.

Cyclic scheduling deals with a set of operations (tasks) that have to be performed an infinite number of times [3]. This approach is also applicable if the number of loop repetitions is large enough. The aim is then to find a periodic schedule with a minimum period. One repetition of the loop, called an *iteration*, can be performed in several periods. Therefore, the schedule is called *overlapped* [4], since the execution of the operations belonging to different iterations can interleave.

Modulo scheduling and *software pipelining* [5] are related terms to *cyclic scheduling*, which are usually used in the compiler community. A common ILP based approach (further called the



(a) (b) (c)

Figure 1: (a) An example of a computation loop lattice wave digital filter (LWDF). (b) Corresponding data dependency graph G. (c) An optimal schedule with $w = 2$.

binary method) uses a binary decision variable x_{it} determining whether a computation of task i starts at clock cycle t. Unfortunately, the size of such an ILP model (corresponding to the time complexity) depends on the period length and therefore it is not suitable for HW architectures with longer input–output latencies of the arithmetic units (e.g. FP32 or HSLA). For example the ILP approach with x_{it} decision variable is used in [6], where the additional objective is to optimize the word–length of the arithmetic units. In [7], the ILP model is extended by automatic processor selection from a library and [4] applies the approach to architectures with interprocessor communication. ILP formulation using the binary method is also employed in [8] to schedule multifunction loop accelerators. A general solution using ILP is shown in [9] but the size of the ILP model is independent of the period length only for a monoprocessor solution.

With respect to these works, our solution leads to a simpler ILP formulation with less integer variables. We propose a method based on ILP to find the optimal periodic schedule. Motivated by FPGA architectures with standard arithmetic units operating with real numbers, we formulated an ILP model for cyclic scheduling of tasks with unit processing time on dedicated sets of parallel identical processors. We call this ILP model the *integer method*, since we use an integer variable to represent the start time of a task while the binary method uses a set of binary variables for the same purpose.

This paper is organized as follows: Section 2 describes the notation and the Basic Cyclic Scheduling problem assuming an unlimited number of processors. The following section presents a formulation of the cyclic scheduling of tasks with unit processing time on dedicated sets of parallel identical processors. Section 4 presents the results demonstrated on benchmarks and shows a comparison with an ILP model using one binary decision variable for each task and each clock cycle within the period. Section 5 concludes the paper.

2 Problem Statement

Operations in an *iterative loop* can be considered as a set of n tasks $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ to be performed K times. One execution of set $\mathcal{T}$ labeled with the integer index $k \geq 1$ is called an *iteration*. The scheduling problem is to find the start time $s_i(k)$ of every occurrence of T_i [3]. Figure 1(a) shows an illustrative example of a simple computation loop (LWDF filter [10]).

Data dependencies of this problem can be modeled by a directed, double weighted graph $G = (\mathcal{T}, \mathcal{E})$. Each task (node in $\mathcal{T}$) is characterized by the processing time p_i. Edge $e_{ij} \in \mathcal{E}$ from the node i to j is weighted by a couple of integer constants l_{ij} and h_{ij}. Length l_{ij} represents the minimal distance in clock cycles from the start time of task T_i to the start time of T_j and is always

greater than zero.

When considering pipelined processors, the processing time p_i represents the time to feed the processor (i.e. new data can be fed into the pipelined processor after p_i clock cycles) and length l_{ij} represents the time of the computation (i.e. the input–output latency). The parameter height h_{ij} specifies the shift of the iteration index (dependence distance) related to the data produced by T_i and consumed by T_j.

Figure 1(b) shows the data dependence graph of the iterative loop shown in Figure 1(a). Processing time p_i is equal to 1 for all arithmetic units. The length l_{ij} corresponds to the input–output latency of the given unit. ADD and MUL unit have 1 and 2 clock cycles input–output latency respectively.

When assuming a *periodic schedule* with *period* w (i.e. the constant repetition time of each task), each edge e_{ij} in graph G represents one precedence relation constraint

$$s_j - s_i \geq l_{ij} - w \cdot h_{ij}, \tag{1}$$

where s_i denotes the start time of task T_i at the first iteration.

The aim of the cyclic scheduling [3] is to find a periodic schedule while minimizing the period w. The scheduling problem is simply solved when the number of processors is the number of processors not limited, i.e. is sufficiently large. The minimal feasible period, given by the *critical circuit* c in graph G, is equal to $\max_{c \in C(G)} \left(\sum_{e_{ij} \in c} l_{ij} \right) / \left(\sum_{e_{ij} \in c} h_{ij} \right)$, where $C(G)$ denotes a set of cycles in G. The critical circuit can be found in polynomial time [3], and we use this value to determine the lower bound of the period in our scheduling problem.

When the *number of processors* m is restricted, the cyclic scheduling problem becomes NP–hard [3]. Unfortunately, in our case the number of processors m is restricted and the processors are grouped into dedicated sets up to their capability to execute specific operations. Three iterations of the optimal schedule to the loop from Figure 1(a) are shown in Figure 1(c). In this case, a hardware architecture with two ADD units and one MUL unit is considered.

3 Integer Linear Programming Formulation

In this section we define an Integer Linear Programming formulation for the problem of cyclic scheduling of tasks with unit processing time on dedicated sets of parallel identical processors $\mathcal{P} = \{\mathcal{P}_1, \ldots \mathcal{P}_d, \ \ldots, \mathcal{P}_D\}$. First we solve a problem for one dedicated set of parallel identical processors (i.e. all tasks in $\mathcal{T}$ are processed on $\mathcal{P}_1 = \mathcal{P}$) which is further generalized in Section 3.2. To define the ILP formulation, we introduce several integer and binary variables.

Let $\hat{s}_i$ be the remainder after division of s_i (the start time of T_i in the first iteration) by w and let $\hat{q}_i$ be the integer part of this division. Then s_i can be expressed as follows

$$s_i = \hat{s}_i + \hat{q}_i \cdot w, \quad \hat{s}_i \in \langle 0, w-1 \rangle, \quad \hat{q}_i \geq 0. \tag{2}$$

This notation divides s_i into $\hat{q}_i$, the index of the *execution period*, and $\hat{s}_i$, the number of clock cycles within the execution period.

Furthermore, let $\hat{x}_{ij}$ be a binary decision variable such that $\hat{x}_{ij} = 1$ if and only if $\hat{s}_i \leq \hat{s}_j$ (i.e. T_i is followed by T_j or both T_i and T_j start at the same time) and $\hat{x}_{ij} = 0$ if and only if $\hat{s}_i > \hat{s}_j$ (i.e. T_j is followed by T_i). And let $\hat{y}_{ij}$ be a binary decision variable such that $\hat{y}_{ij} = 1$ if and only if $\hat{s}_i = \hat{s}_j$ (i.e. tasks T_i and T_j are processed at the same time on different processors).

In the rest of the paper we consider $p_i = 1$ for all $T_i \in \mathcal{T}$. The period w is assumed to be a constant, since multiplication of two decision variables cannot be formulated as a linear inequality.

Therefore, the optimal w is found by iterative calls of the ILP problem. The ILP problem using the above defined variables is:

$$\min \sum_{i=1}^{n} \hat{q}_i \tag{3}$$

subject to:

$$\hat{s}_j + \hat{q}_j \cdot w - \hat{s}_i - \hat{q}_i \cdot w \geq l_{ij} - w \cdot h_{ij}, \qquad \forall (i,j); \exists e_{ij} \in \mathcal{E} \tag{4}$$

$$\hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} + (1-w) \cdot \hat{y}_{ij} \geq 1, \qquad \forall i,j \in \{1,\ldots,n\}; i < j \tag{5}$$

$$\hat{s}_i - \hat{s}_j + w \cdot \hat{x}_{ij} - \hat{y}_{ij} \leq w - 1, \qquad \forall i,j \in \{1,\ldots,n\}; i < j \tag{6}$$

$$-\hat{x}_{ij} + \hat{y}_{ij} \leq 0, \qquad \forall i,j; i < j \text{ and } T_i, T_j \in \mathcal{T} \tag{7}$$

$$\sum_{j=i+1}^{n} \hat{y}_{ij} \leq m - 1, \qquad \forall i \in \{1,\ldots,n-m\} \tag{8}$$

where: $\hat{s}_i \in \langle 0, w-1 \rangle$; $\hat{q}_i \geq 0$; $\hat{s}_i, \hat{q}_i \in \mathbb{Z}$; $\hat{x}_i, \hat{y}_i \in \{0,1\}$

3.1 Constraints of the ILP Model

Constraint (4) is a direct application of precedence constraint (1). Constraints (5), (6), (7) and (8) limit the number of processors used at a given time. The binary decision variables $\hat{x}_{ij}$ and $\hat{y}_{ij}$ define the mutual relation of tasks T_i and T_j ($i \neq j$) within the execution period. Their relation is expressed with constraints (5) and (6). There are three feasible combinations:

1. When $\hat{x}_{ij} = 0$ and $\hat{y}_{ij} = 0$, constraint (6) is eliminated in effect. Constraint (5) reduces to $\hat{s}_j + 1 \leq \hat{s}_i$, i.e. T_j is followed by T_i within the execution period.
2. When $\hat{x}_{ij} = 1$ and $\hat{y}_{ij} = 0$, constraint (5) is eliminated in effect. Constraint (6) reduces to $\hat{s}_i + 1 \leq \hat{s}_j$, i.e. T_i is followed by T_j within the execution period.
3. When $\hat{x}_{ij} = 1$ and $\hat{y}_{ij} = 1$, constraints (5) and (6) are equivalent to $\hat{s}_i = \hat{s}_j$, i.e. T_i and T_j are scheduled at the same time within the execution period.
4. Combination $\hat{x}_{ij} = 0$ and $\hat{y}_{ij} = 1$ is not feasible due to constraint (7).

The number of available processors is limited using the variable $\hat{y}_{ij}$. When $\hat{y}_{ij} = 1$, there is a resource conflict between tasks T_i, T_j and they can not be scheduled on the same processor. Since we consider the unit processing time of the tasks, this relation between the tasks expressed with $\hat{y}_{ij}$ is symmetric ($\hat{y}_{ij} = \hat{y}_{ji}$) and transitive (if $\hat{y}_{ij} = 1$ and $\hat{y}_{jk} = 1$ then $\hat{y}_{ik} = 1$). The relation can be expressed by graph G^y where the nodes represent the tasks in $\mathcal{T}$. There is an edge between T_i and T_j iff $\hat{y}_{ij} = 1$. Due to the symmetry and transitivity, the graph G^y consists of disjoint complete subgraphs H_u^y, such that $G^y = \{H_1^y, \ldots, H_u^y, \ldots, H_U^y\}$, where $U \leq w$. Each H_u^y corresponds to the set of tasks executed at the same time. Then for each H_u^y, it holds that the number of nodes in H_u^y is equal to the chromatic number $\chi(H_u^y)$ which is equal to $\Delta(H_u^y) + 1$, where $\Delta(H_u^y)$ is the maximum degree in H_u^y [11]. In our case $\Delta(H_u^y) = \sum_{j=1}^{n} \hat{y}_{ij}$ where T_i is a node of H_u^y.

In order to limit the maximal number of processors used, we limit the number of nodes in the biggest H_u^y. It can be expressed as a condition $\sum_{j=1}^{n} \hat{y}_{ij} \leq m-1, \forall i \in \{1,\ldots,n\}$ and can be further simplified to the form of constraint (8) using the following lemma

Lemma 1. *Constraint* $\sum_{j=1}^{n} \hat{y}_{ij} \leq m-1, \forall i \in \langle 1, n \rangle$ *is equivalent to constraint* $\sum_{j=i+1}^{n} \hat{y}_{ij} \leq m-1$, $\forall i \in \langle 1, n-m \rangle$, *in the ILP model.*

Proof. $\sum_{j=1}^{n} \hat{y}_{ij} = \sum_{j=1}^{i} \hat{y}_{ij} + \sum_{j=i+1}^{n} \hat{y}_{ij}$. By induction we show that the first term of this addition can be omitted.

$i = 1$: $\sum_{j=1}^{i} \hat{y}_{ij} = \hat{y}_{11} = 0$

$i = 2$: $\sum_{j=1}^{i} \hat{y}_{ij} = \hat{y}_{21} + \hat{y}_{22} = \hat{y}_{21}$. If $\hat{y}_{21} = 0$ then $\hat{y}_{21}$ has no effect on constraint (8). If $\hat{y}_{21} = 1$ then nodes T_1 and T_2 belong to the same H_u^y and the number of nodes in H_u^y is already limited in constraint (8) for $i = 1$.

$i = k$: $\sum_{j=1}^{i} \hat{y}_{ij} = \hat{y}_{k1} + \hat{y}_{k2} + \ldots + \hat{y}_{kk} = \hat{y}_{k1} + \ldots + \hat{y}_{k(k-1)}$. If $\hat{y}_{k1} + \ldots + \hat{y}_{k(k-1)} = 0$ then $\hat{y}_{k1}, \ldots, \hat{y}_{k(k-1)}$ have no effect on constraint (8). If $\hat{y}_{k1} + \ldots + \hat{y}_{k(k-1)} > 0$ then at lest one node T_v ; $v \in \langle 1, k-1 \rangle$ belongs to the same H_u^y. Thereafter, the number of nodes in H_u^y is already limited by constraint (8) for some $i = v$.

Finally, constraint (8) can be formulated for $i \in \langle 1, n-m \rangle$, since $\sum_{j=n-m+1}^{n} \hat{y}_{ij} \leq m-1$ is always satisfied. □

Please notice that we do not directly assign the tasks to the processors but we only restrict the number of tasks executed at the same time. The above shown ILP formulation requires $2 \cdot n$ integer and $n^2 - n$ binary variables constrained in $n_e + 3/2 \cdot (n^2 - n) + n - m$ constraints. It is obvious that the size of the ILP model depends only on the number of tasks n, the number of edges n_e and the number of processors m.

3.2 Dedicated Sets of Parallel Identical Processors

The above mentioned formulation of the processor constraints allows one to generalize the approach to the problem with dedicated sets of parallel identical processors where each task is assigned to a dedicated set of parallel identical processors $\mathcal{P}_d$ such that $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_d, \ldots, \mathcal{P}_D\}$ where D is the number of these sets. Then $m_d = |\mathcal{P}_d|$ denotes the number of processors in set $\mathcal{P}_d$ and m, the number of processors, is equal to $\sum_{d=1}^{D} m_d$. For example, in one of the benchmarks in Section 4.1 we assume to have 3 dedicated sets (one for addition, one for multiplication and one for division). Furthermore, n_d denotes the number of tasks assigned to the set $\mathcal{P}_d$ and n, the number of tasks, is equal to $\sum_{d=1}^{D} n_d$.

The tasks assigned to different processors are not conflicting, i.e. they do not impose any restriction with respect to the feasibility of the schedule. Therefore, constraints (5), (6), (7) and (8) are generated separately for each set $\mathcal{P}_d$ with m_d processors and n_d tasks. On the other hand, constraint (4) remains the same, since it represents the precedence constraints among the tasks that may run on different processors.

3.3 Objective Function

We recall that the goal of cyclic scheduling is to find a feasible schedule with the minimal period w. Therefore, w is not constant as we assumed in the ILP formulation above and it is a positive integer value. Period w^*, the shortest period resulting in a feasible schedule, is constrained by its lower bound and its upper bound [12]. The optimal period w^* can be found iteratively by formulating one ILP model for each iteration. These iterative calls of ILP do not need to be performed for all w between the lower bound and the upper bound, but the interval bisection method or simple incrementing of the period can be used, until w^* is found.

Using the ILP formulation we are able to test the schedule feasibility for a given value of w. In addition, we minimize the secondary criterion formulated using the objective function of ILP.

One of the simplest objectives is to minimize the iteration overlap by objective function (3) as is assumed in this paper. Another criterion is the minimization of the iteration length or minimization of the data storage units for transfers among the tasks [13].

4 Results

The presented scheduling technique was implemented and run on an AMD Opteron at 2.2 GHz using the ILP solver tool CPLEX [14]. The complexity of integer linear programs can be estimated by using the number of integer variables in the ILP model, but each change of the problem instance may lead to a significant change of the algorithm computation time.

4.1 Standard Benchmarks

We compare the result of the binary method (e.g. [4, 8, 6]) and the integer method (shown in Section 3) on standard benchmarks: the second order wave digital filter (WDF) [15], the Jaumann filter (JAUMANN) [16], the recursive least squares filter (RLS) [12], the seventh-order biquadratic IIR filter (IIR7) [17] and the fifth–order wave digital elliptic filter (ELLIPTIC) [18].

					integer method			binary method		
Benchmark	n	arithmetic units	q_{max}	$w*$	#var	*#constr*	*CPU time*	*#var*	*#constr*	*CPU time*
WDF	8	2ADD ($l_{ij}=2$), 2MUL ($l_{ij}=3$)	3	9	48	65	**0.016**	80	39	**0.063**
	8	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	3	29	48	65	**0.016**	240	79	**0.047**
JAUMANN	17	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	2	58	202	293	**0.016**	1003	161	**0.063**
	17	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$)	2	82	202	293	**0.016**	1411	209	**0.688**
RLS	26	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$), 2DIV ($l_{ij}=2$)	4	26	320	454	**0.047**	702	135	**0.172**
	26	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$), 2DIV ($l_{ij}=28$)	2	74	320	454	**0.031**	1950	279	**0.953**
IIR	29	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	8	20	450	657	**0.125**	609	134	**0.234**
	29	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$)	7	30	450	657	**0.063**	899	133	**0.156**
ELLIPTIC	34	2ADD ($l_{ij}=2$), 2MUL ($l_{ij}=3$)	3	29	774	1147	**0.063**	1020	150	**0.313**
	34	2ADD ($l_{ij}=9$), 2MUL ($l_{ij}=2$)	2	96	774	1147	**0.063**	3298	284	**576.250**
	34	2ADD ($l_{ij}=11$), 2MUL ($l_{ij}=8$)	2	134	774	1147	**0.063**	4590	360	**1.406**

Table 1: Benchmarks

Experiments are summarized in Table 1 where n denotes the number of tasks, $\hat{q}_i$ denotes the upper bound of q_{max} given *a priory*. The parameters of dedicated sets of processors including the number of processors in the set, function of the unit and corresponding input–output latency l_{ij} in brackets are shown in the column *arithmetic units*. In our experiments, we have primarily considered the parameters of the FP32 and HSLA libraries [1, 2]. The algorithm results are given by the shortest period resulting in a feasible schedule w^*. The parameters of the ILP models (integer and binary) in Table 1 are the number of ILP variables *#var*, the number of ILP constraints *#constr* and the *CPU time*. The CPU time is the time required to compute the optimal solution, given as a sum of iterative calls of the ILP solver.

4.2 Randomly Generated Instances

To compare the time complexity of the integer method with the one of the binary method thoroughly, we evaluate the average CPU times for randomly generated instances.

The randomly generated graph G consists of $\lceil 2/3 \cdot n \rceil$ edges of height $h_{ij} = 0$ and n edges of height $h_{ij} > 0$. The height $h_{ij} > 0$ is chosen from a uniform distribution on the interval $\langle 1, 3 \rangle$ (and rounded toward the nearest integer). The out degree of the nodes in the generated graph G is less then or equal to 4. All tasks (corresponding to the nodes) were associated with one set consisting of two parallel identical processors. To demonstrate the influence of the precedence delay on the

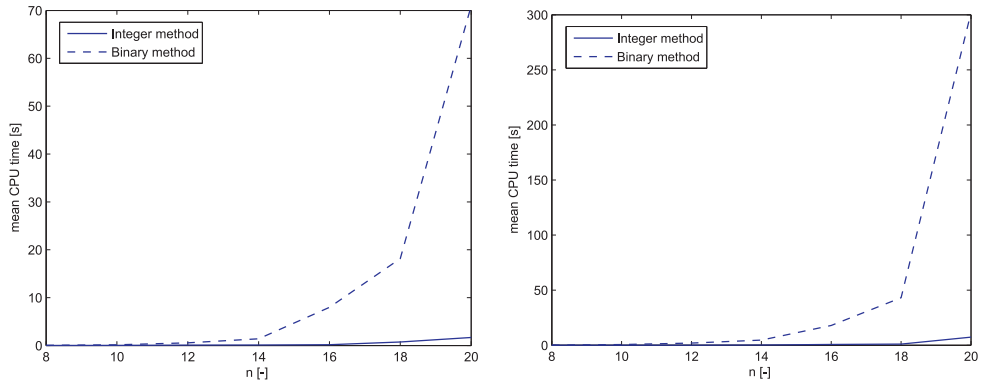


Figure 2: Comparison of integer and binary method.

CPU time, we performed two benchmarks, one for $l_{ij} = 6$ (see Figure 2(a)) and one for $l_{ij} = 9$ (see Figure 2(b)).

For each number of tasks n, 500 test instances have been randomly generated. The mean *CPU time* in dependence on the number of nodes (tasks) n is shown in Figures 2(a) and (b). The results show considerably lower time requirements for the integer method. From the comparison between figures 2(a) and 2(b), it follows that for instances with greater precedence delays ($l_{ij} = 9$), the two methods differ even more.

5 Conclusions

This paper presents an ILP–based cyclic scheduling on dedicated sets of parallel identical processors. In this paper, we have focused on problems with unit processing time of tasks which is common in most FPGA architectures. Experimental results show that our model is more effective for problems with a longer period than the commonly used ILP model where the number of variables is independent of the period length (the binary method) [4, 8, 6]. The time complexity of the presented ILP model can be further optimized by using the *elimination of redundant processor constraints* presented in [13].

6 Acknowledgement

This work was supported by the Ministry of Education of the Czech Republic under Project 2C06017 and Research Programme MSM6840770038.

References

[1] Celoxica Ltd. (2004), Platform Developers Kit: Pipelined Floating-point Library Manual, http://www.celoxica.com.

[2] R. Matoušek, M. Tichý, Z. Pohl, J. Kadlec and C. Softley (2002), Logarithmic number system and floating-point arithmetics on FPGA, *International Conference on Field-Programmable Logic and Applications (FPL '02)*, 627 – 636.

[3] C. Hanen and A. Munier (1995), A study of the cyclic scheduling problem on parallel processors, *Discrete Applied Mathematics* **57**, 167 – 192.

[4] S. L. Sindorf and S. H. Gerez (2000), An integer linear programming approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays, *PROGRESS 2000 Workshop on Embedded Systems*, Utrecht, The Netherlands.

[5] B. R. Rau and C. D. Glaeser (1981), Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing, *MICRO 14: Proceedings of the 14th annual workshop on Microprogramming*, Piscataway, USA, 183 – 198.

[6] K.-I. Kum and W. Sung (2001), Combined word-length optimization and high-level synthesis of digital signal processing systems, *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems* **20**, 921 – 930.

[7] K. Ito, E. Lucke and K. Parhi (1999), ILP based cost-optimal DSP synthesis with module selection and data format conversion, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **6**, 582 – 594.

[8] K. Fan, M. Kublur, H. Park and S. Mahlke (2006), Increasing hardware efficiency with multi-function loop accelerators, *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, 276 – 281.

[9] D. Fimmel and J. Müller (2001), Optimal software pipelining under resource constraints, *Journal of Foundations of Computer Science* **12**, 697 – 718.

[10] M. Vesterbacka, K. Palmkvist, P. Sandberg and L. Wanhammar (1994), Implementation of fast DSP algorithms using bit-serial arithmetic, *National Conf. on Electronic Design Automation*, Kista, Stockholm.

[11] D. B. West (2001), *Introduction to Graph Theory*, second edition, Prentice Hall.

[12] P. Šůcha, Z. Pohl and Z. Hanzálek (2004), Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit, *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*, 404 – 412.

[13] P. Šůcha, Z. Hanzálek, A. Heřmánek, and J. Schier (2007), Scheduling of iterative algorithms with matrix operations for efficient FPGA design–implementation of finite interval constant modulus algorithm, *Journal The Journal of VLSI Signal Processing* **46**, 35 – 53.

[14] ILOG Inc. (2005), *CPLEX Version 9.1*, http://www.ilog.com/products/cplex/.

[15] A. Fettweis (1986), Wave digital filters: theory and practice, *Proceedings of the IEEE* **74**, 270 – 327.

[16] S. M. Heemstra, S. H. Gerez and O. E. Herrmann (1992), Fast prototyping of datapath-intensive architectures, *IEEE Transactions on Circuits and Systems I* **39**, 351 – 364.

[17] J. M. Rabaey, C. Chu, P. Hoang and M. Potkonjak (1991), Fast prototyping of datapath-intensive architectures, *IEEE Des. Test* **8**, 40 – 51.

[18] E. Bonsma and S. Gerez (1997), A genetic approach to the overlapped scheduling of iterative data-flow graphs for target architectures with communication delays, *ProRISC Workshop on Circuits, Systems and Signal Processing.*

Evaluating Online Scheduling Techniques in Uncertain Environments

Michael Thomas, Helena Szczerbicka
FG Simulation & Modellierung, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany
{mit, hsz}@sim.uni-hannover.de

Online scheduling (also known as dynamic scheduling) refers to the real-time coordination of operation processing while new operations are continuously arriving. The objective of this paper is the study of different online scheduling techniques in a dynamic, stochastic job shop scheduling environment. We present a quantitative study of the quality of dispatching and optimizing scheduling techniques under varying uncertainty and utilization. Therein, we also consider a regret-based algorithm adopted from Bent and Van Hentenryck [1] in a setting with stochastic processing times and analyze its performance. To date, results in such a dynamic setting are lacking, while it has been shown to outperform other heuristic optimization methods in online packet scheduling with processing times known a priori.

Keywords: Online Scheduling, Job-Shop Scheduling, Heuristic Search.

1 Introduction

In recent years, automated scheduling techniques have received increasing attention. Such techniques have great impact in reducing the operational cost and in increasing the autonomy of complex manufacturing systems. Contrary to static job shop scheduling problems, the data to be processed is not known in advance but rather revealed over time. The task of the scheduling system is to decide which request has to be served next while minimizing a given objective function, e. g. the maximum or weighted average lateness.

While the notion of dynamic scheduling is frequently used in computer architectures as the technique for speeding up execution, this paper refers to dynamic scheduling (also called online scheduling) in a dynamic, stochastic job shop setting. That is a job shop with stochastic operation processing times and jobs, composed of a sequence of (atomic) operations, that arrive over the course of time. We allow precedence delays, transition times between different operations on a resource and—unlike most publications on online scheduling—an operation to require multiple resources to be processed. However, in operation processing resource-sharing and preemption are not allowed.

In this setting, we compare the quality of schedules obtained from (a) a dispatch rule, i. e. sequentialization by rules, and (b) sequentialization by optimization algorithms, with respect to the variances of the random variates in the problem instance. Furthermore, we analyze the performance of the Regret approach proposed by Bent et al. [1, 2] since it lets open how and whether sampling of future events can still improve a combinatorial optimization algorithm in a scenario with stochastic timings.

The remainder of the paper is structured as follows. In Section 2, related work on online scheduling and the dynamic aspects of scheduling problems is briefly discussed. Section 3 introduces the dynamic job shop scheduling problem and our framework. In Section 4, the scheduling algorithms are described. Section 5 contains experiments and results, and Section 6 concludes the paper.

2 Related Work

For a long time, research mostly concentrated on scheduling as a static problem. In recent years, this perspective shifted towards the dynamic aspects and appropriate scheduling techniques. These dynamic scheduling techniques can be roughly distinguished into the following three branches that we will summarize briefly.

The first approach to dynamic scheduling is dispatching with rules. This approach is very common and well suited for systems where short response times are required and the future is uncertain, such as operation systems or network routers. A variety of rules has been developed (for a survey, see e. g. [9]). However, rule performance depends on the problem instance, thus the challenge is to determine which rules perform best [7]. This issue has been addressed with dynamic rule selection using iterative learning and heuristics [6, 12]. Nevertheless, these schedulers generally lack the ability to plan into the future what renders them suboptimal in the presence of stochastic prediction models.

The second approach to dynamic scheduling derives from the static methods. It continuously recomputes the schedule via combinatorial optimization to reflect changes in the environment, like newly arriving jobs, etc. That is, periodically or upon significant events, a snapshot of the system is taken and processed by algorithms for static scheduling, see e. g. [5, 3]. This approach has been extended by Sun et al. [10] or Smith [11] to distinguish severity among the changes: scheduling is divided into a planning phase, that recomputes the whole schedule, and an execution phase in which the current schedule is repaired by a light-weight scheduler. Approaches of the second kind are well suited in predictable environments with a time-scale large enough for schedule computation.

The third approach to dynamic scheduling is stochastic optimization, in which schedule (re-) computation is done with respect to a stochastic model of the future. While this approach seems similar to the second one, it explicitly integrates future events and adjusts the schedule accordingly. Therefore most algorithms sample future events from a stochastic (or empirical) model for each possible scheduling decision. Bent and Van Hentenryck propose an algorithm that enables the evaluation of every decision on all samples, that is without distributing the samples among them [1]. In [4], Chang et al. formulate a partially observable Markov decision process for simplified scheduling problems (in particular, the authors assume the absence of precedence constraints) using a hidden Markov model for the arrival process.

Still, an issue pointed out by Montana in [8] remains unsolved. In the presence of limited computational power and varying processing times, is it still reasonable to compute an optimized schedule? Or does, under these conditions, dispatching perform equally well with less costs? Hence, in the following we will analyze under which conditions combinatorial optimization (with and without sampling of the future) reasonably outperforms dispatching rules and how sampling may improve the optimization results.

3 The Job Shop Framework

For the sake of clarity, we first formally define a dynamic stochastic job shop problem and then describe our framework.

3.1 Formal Definition

An instance of the dynamic stochastic scheduling problem, in the following also referred to as *scenario*, consists of basically three types of entities: *resources*, *jobs* and *operations*. Jobs are temporal entities that continuously arrive into the system and specify a sequence of operations. These operations have to be processed sequentially in the given order to complete the job. Resources

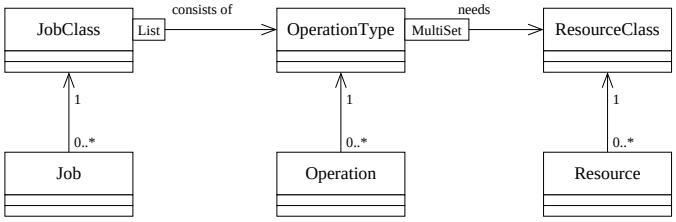


Figure 1: Structure diagram of a scenario. Arrows indicate associations.

are permanent entities needed for operation processing. Each resource can process at most one operation at a time. There are resource-dependent transition times between different operations; preemption is not allowed. Resources are partitioned into *resource classes*, jobs into *job classes* and operations into *operation types*, so that all entities in each class and type share the same properties:

Resource Classes. A resource class specifies transition (or setup) times between operation types and the number of resources in that class.

Operation Types. An operation type specifies the number of resources per resource class needed for its execution and a probability distribution of its execution times.

Job Classes. A job class specifies the sequence of operation types to be processed in order to complete the job, a sequence of precedence delays, a fixed weight and probability distributions for the interarrival and execution time (the time between a job's release and its deadline).

Figure 1 shows a diagram of the resulting structure. To define the dynamics of a scenario, consider a finite time horizon $H = [0, h]$, $h \in \mathbb{R}$. A configuration of a dynamic scheduling instance at time t, $0 \leq t \leq h$, can be described as a quadruple $I(t) = (R, J, O(t), f(t))$, where $R = (R_1, \ldots, R_n)$ denotes a vector of available resources per resource class, $J = (J_k)_{1 \leq k \leq m}$ is the vector of job arrival processes with one component for each class, $O(t)$ is the set of arrived but unserved operations and $f\colon R \to \mathbb{R}_{\geq 0}$ is a mapping that associates with each resource $r \in R$ its latest release time $\leq t$ (i. e. the latest point in time $\leq t$ that an operation using resource r completed). Further on, we have to distinguish operations in $O(t)$ into schedulable and unschedulable operations, since arriving jobs introduce a sequence operations whose members have to be processed sequentially with respect to the given precedence delays. We will refer to the set of schedulable operations at time t as $O_{\text{sched}}(t)$ and define the release time of operation o as $a_o = \min\{t \mid o \in O_{\text{sched}}(t)\}$; the associated job will be denoted j_o. We will also refer to J as the set of all jobs arriving in $[0, H]$ and O as the set of all operations, respectively.

A solution to an online scheduling problem is a mapping $s\colon R \times H \to \bigcup_{t \in H} O_{\text{sched}}(t)$, called *schedule*, such that $s(t) = o$ if operation $o \in O(t)$ is scheduled at time t, and $s(t) =$ undefined, otherwise.

The goal of an online scheduler is to compute $s(t)$ with the knowledge of $I(t)$ while minimizing a given objective function. In this paper, we fix this function to the weighted average lateness $\frac{1}{|J|} \sum_{j \in J} (c_j - d_j)$, where c_j denote the completion time of the last operation and d_j the due date of job j (other examples for an objective function are e. g. the maximum lateness $\max_{j \in J}\{c_j - d_j\}$).

3.2 Framework

We set up a modular framework, to be able to evaluate all online scheduling techniques under exactly the same conditions. Basically, this framework is composed of four components: the simulator driving the discrete event simulation, a scenario describing an instance of a dynamic, stochastic

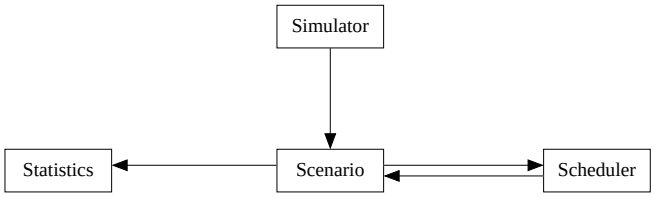


Figure 2: The block diagram of the framework. Arrows indicate control flow during execution.

```
1  function chooseOperation():
2    RC ← availableResources()
3    opList ← computePriorityList(O_sched(t))
4    for (i ← 1 to |O_sched(t)|) do
5      if (neededResources(opList[i]) ≤ RC) then
6        schedule(opList[i])
7        RC ← RC− resourceClasses(opList[i])
8      end if
9    end do
10
11
12
```

Figure 3: The generic scheduling algorithm.

```
1  function computePriorityList(O_sched(t)):
2    for (o ∈ O(t)) do
3      r(o) ← 0
4    end do
5    for (i ← 1 to k) do
6      F ← sampleFutureEvents()
7      s ← computeScheduleGA(O(t) ∪ F)
8      for (o ∈ O(t))
9        r(o) ← r(o) + eval(s) + regret(s, o)
10     end do
11   end do
12   return sort(O_sched(t), r)
```

Figure 4: The Regret priority computation.

job shop, a scheduler reacting to the events in the scenario and a data acquisition module tracking the desired statistics. Figure 2 shows the control-flow in the framework. In a preprocessing step, all stochastic times are drawn from their respective distributions to ensure an equal amount of variance in the each simulation. In particular, this enables accounting differences directly to the scheduling technique. Then, for each scheduler, the simulator processes the events generated in the preprocessing step in increasing timestamp order, while events, on their part, trigger the scheduler and log the statistics afterwards.

We have validated the framework using the scenarios and dispatch scheduler from [8] and obtained the same results within a range of 5% that we credit to missing details in the framework description.

4 Scheduling Algorithms

This section introduces the online scheduling algorithms considered in this paper. The algorithms were selected as representatives for the different scheduling techniques mentioned in Section 2, and chosen for two reasons. First, they provide reasonable performance compared to other schedulers of the same type. And second, they are relatively straightforward implementations and have already been treated in several papers.

Upon event occurrences, all scheduling algorithms are triggered by a call to the function chooseOperation() shown in Figure 3. There, the function availableResources() returns a vector containing the number of free resources per class, whereas neededResources(o) returns the number of resources per class required to execute operation o. The function schedule(o) schedules operation o and, at last, the function computePriorityList(O) returns a list of operations in O, ordered according to the sequence in which they shall be scheduled. This is the only function that the following schedulers differ in.

4.1 Dispatch-Rule Scheduler

Dispatch-rule schedulers do not look ahead in terms of planning schedules. Their general approach is to rank each operation according to a given dispatch rule and execute the operation that maximizes (or minimizes) that score as soon as all required resources can be allocated.

In this paper, we consider two schedulers of this kind: one implementing a *first-come-first-served (FCFS)* strategy, the other one implementing an modified *least-slack (LS)* strategy. For the FCFS scheduler, computePriorityList($O_{\text{sched}}(t)$) returns the operations ordered by ascending release times a_o, $o \in O_{\text{sched}}(t)$; for the LS scheduler, the list is ordered ascending according to

$$\pi(o) = \frac{s_{\text{initial}}(j_o)}{s_{\text{now}}(j_o)} \cdot w_{j_o} \cdot \big(\mathrm{P}_{assign}(o) - \mathrm{P}_{wait}(o)\big),$$

where o is an operation, w_j the weight of job j, $s_{\text{initial}}(j)$ its initial and $s_{\text{now}}(j)$ its current slack time[1], $\mathrm{P}_{assign}(o)$ the probability that j_o fails to complete on time if o is scheduled now and $\mathrm{P}_{wait}(o)$ the probability that j_o fails to complete on time if o is not scheduled now. That is, $\pi(o)$ equals the ratio of j_o's initial slack time to its current slack time multiplied with j_o's weight and the probability loss of not being late. The estimation of $\mathrm{P}_{assign}(o)$ and $\mathrm{P}_{wait}(o)$ has been done as in [8].

4.2 Optimizing Scheduler

Optimizing schedulers treat the dynamic job shop as a static problem. They compute a schedule for all arrived but unserved operations using combinatorial optimization algorithms without consideration of future events and times. To this end, the unknown stochastic processing times are substituted with their mean values.

The optimization algorithm applied is the genetic algorithm described in [3] using the permutation with repetition encoding and the hybrid Giffler-Thompson algorithm with $\delta = 0.5$ for the decoding step. The population size is kept constant at $\min\{|T(t)|!, 100\}$. We use elitism for 20% of the population and generalized order crossover for the remaining 80%—this crossover operator tends to respect the relative order of operations. Mutation is applied with a probability of 1%. The optimization criteria employed are estimated maximum and weighted average lateness. The genetic algorithm is initialized with chromosomes from the dispatch schedulers above, iterated for 30 generations, and finally, returns the operations in $O_{\text{sched}}(t)$ ordered according to best chromosome found.

4.3 Stochastic Optimizing Scheduler

A stochastic optimizing scheduler works similar to an optimizing scheduler but additionally incorporates sampled future job arrivals and processing times from a stochastic (or empirical) model of the job shop. This way, the schedules are expected to be more robust to strongly deviating processing times.

We have developed an algorithm which is an adoption of the Regret approach in [1]. The Regret approach assumes the existence of a function regret() that bounds the costs for scheduling a suboptimal operation instead of an optimal one. This bears the advantage that, for every sample, each operation can be rated according to its regret: eventually the operation minimizing the overall regret is scheduled on a broader information basis. The regret function, on the other hand, has to be computable in $\mathcal{O}(f_{\text{Sched}})$, where f_{Sched} refers to the time complexity of the underlying scheduler. Otherwise the algorithms' time complexity would increase beyond $\mathcal{O}(f_{\text{Sched}}/|T(t)|)$.

The pseudocode of the Regret algorithm is given in Figure 4, where computeScheduleGA(O) invokes the optimizing GA scheduler for the operations in O, eval(s) denotes the evaluation of the

[1]Slack time is defined as the time to deadline less the remaining processing and necessary delay times.

schedule s according to the chosen optimization criterion and $\textsc{sort}(O, r)$ returns a list of elements $o \in O$ in ascending order of the value $r(o)$.

The main difference between our algorithm and the original is a specialized seeding mechanism that includes the best 20% chromosomes from the previous optimization. We observed a reduction of the search costs for solutions of the same quality. This is consistent with results from [13]. The regret of an operation has been estimated very roughly by assuming its global left shift (including its unfinished predecessors). Hence, the regret $r(o)$ of an operation o is given as

$$r(o) = p_o + \sum_{o' \in I} p_{o'},$$

where p_o denotes the (remaining) processing time of operation o and I denotes the set of all unfinished predecessor operations of o in job j_o.

5 Experimentation

In order to evaluate the performance of the schedulers introduced in Section 4, we have generated random problems with varying system utilization and variance in the stochastic variates. The generation process is described in Section 5.1, whereas the results are discussed in Section 5.2.

5.1 Setup

We found that most randomly selected problems are either too easy or instable. To generate "difficult" problems in the sense that they are close to being schedulable on time but to experience lateness in trivial first-come-first-served case, we proceeded in the following way.

Given the number of job classes n and the number of operations per job m, m resource classes $\{1, \ldots, m\}$ with n resources are generated. Afterwards, n sets of m operation types $O_j = \{o_{j,r} \,|\, 1 \leq r \leq m\}$, $1 \leq j \leq n$, are created, where $o_{j,r}$ uses one resource of class r. Additional resource usage is randomly associated by drawing the number of additional resources from a geom($\frac{3}{m}$)-distribution and their classes from a unif$\{1, \ldots, m\}$-distribution. The operations in O_j are then assigned to job j in a random permutation. This ensures a mean $\frac{m}{3}$ operations with more than one resource—a moderate degree of competition among the job classes.

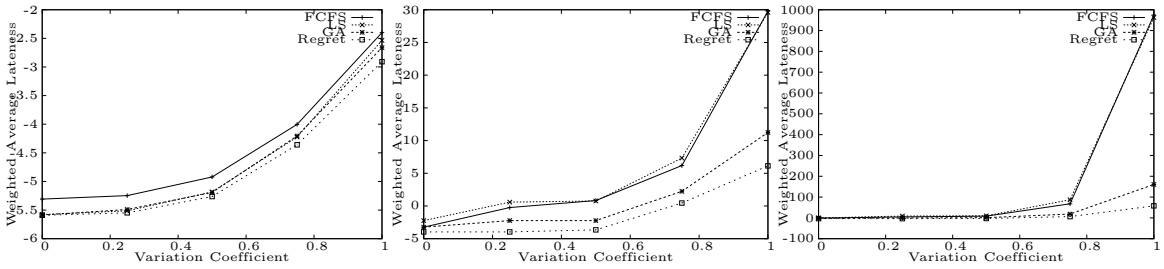
In our simulations, we assumed fixed transition times and precedence delays, exponentially distributed interarrival times as well as truncated normal distributions for the processing times and due dates. The number of job classes was fixed to 4 and the number of operations per job class to 6. The mean processing times were uniformly chosen from the interval $[3.0, 10.0]$, precedence and transition times were fixed to one third of its mean. Due dates were set to three times a job's mean processing time (including delays). Both the variances and arrival rates were varied throughout the simulations.

We created four branches of scenarios with increasing maximum resource utilization $\rho \in \{0.50, 0.65, 0.78, 0.95\}$ and increasing processing time variation coefficient $c = \frac{\sigma}{\mu} \in [0, 1]$.[2] The simulation length was chosen to allow approximately 10 000 job arrivals and runs were repeated independently 10 times. Hence, the size of the 95% confidence interval I for the lateness can be estimated by $|I| \approx \frac{c+\Delta t}{100} \cdot \mu$, where Δt is the mean time between arrivals. The number of samples for the Regret scheduler was adaptively set to $(1 + 5 \cdot c)$ with 10 generations per sample in the genetic algorithm.

5.2 Results and Discussion

In the following, the results conducted from the experiments will be summarized. Figure 5 and Table 1 provide exemplary data.

[2] μ denotes the mean of a processing time and σ its standard deviation.



(a) $\rho = 0.6530$ (b) $\rho = 0.7862$ (c) $\rho = 0.9359$

Figure 5: Weighted average lateness versus the estimated variation coefficient $c = \frac{\sigma}{\mu}$ at resource utilization ρ.

	Mean computation time per schedule					
	$\rho = 0.6530$			$\rho = 0.9359$		
Scheduler	$c = 0$	$c = 0.5$	$c = 1$	$c = 0$	$c = 0.5$	$c = 1$
GA	$3.206 \cdot 10^{-3}$	$3.894 \cdot 10^{-3}$	$8.265 \cdot 10^{-3}$	$3.908 \cdot 10^{-2}$	$4.366 \cdot 10^{-2}$	$5.467 \cdot 10^{-1}$
Regret	$9.858 \cdot 10^{-3}$	$1.120 \cdot 10^{-2}$	$2.275 \cdot 10^{-2}$	$8.733 \cdot 10^{-2}$	$7.201 \cdot 10^{-2}$	$8.834 \cdot 10^{-1}$

Table 1: Mean computation time per schedule.

Averaged over all our simulations, the weighted average lateness performance criterion significantly outperformed maximum lateness in more than half of the runs (to a confidence of 95%, with lower differences in high utilization scenarios). Nevertheless did the schedulers using different optimization criteria exhibit similar behavior, hence the results concerning the maximum lateness criterion are omitted.

For the cases with a maximum utilization $\rho < 0.6$, both dispatching rules and optimization algorithms perform equally well for all variations of c. This is almost surely due to the lack of resource competition.

In the more interesting case of high utilization, $0.6 \leq \rho < 1.0$, optimization of the schedule bears a significant improvement of the computed schedules. We hypothesize that the improvement grows quadratically in the variation coefficient c, as the queue length in $M/G/1$ models in steady state is $2\frac{\rho^2}{1-\rho}(1+\sigma^2\mu^2) = k(\rho)\cdot(1+c^2\mu^4)$, where $k(\rho)$ is a constant depending on ρ. Indeed, this hypothesis also provides an explanation for the rapid decay in the quality of the solutions computed by the dispatching schedulers with growing utilization. Thus, we conclude that despite (polynomially growing) runtimes[3] of the GA and Regret algorithms, their application is feasible because even under strong time constraints. For example, an optimization constrainted to a runtime of 1ms seeded with results from dispatching algorithms showed mean improvements of 10% ($c = 0.0$) to 30% ($c = 1.0$) in another series of simulations.

Additionally the plain genetic algorithm is still outperformed by the Regret scheduling algorithm despite its coarse regret function (global left-shift) in the more relaxed setting with stochastic timings and precedence constraints. Actually this improvement grows with increasing variance of the processing times. This can be accounted to the structure and growing size of the solution space [13]. The GA scheduler cannot sufficiently search that space in the limited amount of time anymore and might get stuck in suboptimal solutions. The Regret scheduler on the other hand searches a different but still structurally similar search space for every sample. These repetitions allow to escape suboptimal solutions and improve the optimization results. However, this statement remains true only in the case of an adaptive number of samples. Otherwise misjudgments become

[3]The time complexity of our hybrid Giffler-Thompson algorithm is in $\mathcal{O}\left(c_R \cdot T(t)^2 \log T(t)\right)$, where c_R is constant that depends on R. The runtime of the genetic algorithm remains constant beyond a certain problem size.

likely and the performance of the Regret scheduler degrades. To test this hypothesis, we fixed the number of samples while increasing c up to 2. The resulting performance degraded stepwise, even beyond the performance of the GA-scheduler.

6 Conclusion

We compared schedule quality obtained from dispatch and optimization schedulers under varying coefficient of variation and resource utilization. From our results, it can be concluded that the application of an optimizing scheduler is feasible as soon as any notable resource competition appears, particularly as the optimization algorithms can be adapted to to the system's time requirements or system-specialized search heuristics. Even in our setting with loose due dates, this limit was reached with a fairly low utilization of $\rho = 0.6$.

Furthermore, we showed that in job shop problems with stochastic processing times, a regret-based scheduler approach retains the ability to improve over plain optimization algorithms, regardless of the utilization or variation (in the case of an adaptive number of samples). According to simulation results only briefly mentioned herein, we account this to the structure of the search space (cp. Watson et al. [13]). We suspect that this observation only holds for the case of moderate resource competition. An optimized resource allocation might increase a schedule's accuracy beyond the gain of the Regret approach if the difficulty of resource allocation increased further.

Currently a study on this supposition is subject to our research. In further work we want to investigate whether it is possible to combine advantages of the scheduling techniques using a Markov Decision Process as it has been done in [4] with different dispatching rules.

References

[1] R. Bent, P. Van Hentenryck (2004), Regrets Only! Online Stochastic Optimization under Time Constraints, *Proc. of the 19th National Conference on Artificial Intelligence*, 501 – 506.

[2] R. Bent, P. Van Hentenryck, E. Upfal (to appear), Online Stochastic Optimization Under Time Constraints,

[3] C. Bierwirth, D.C. Mattfeld (1999), Production Scheduling and Rescheduling with Genetic Algorithms, *Evolutionary Computing* **7**(1), 1 –17 .

[4] H.S. Chang, R. Givan, E.K.P. Chong (2000), On-line Scheduling via Sampling, in *Artificial Intelligence Planning Systems*, 62 – 71.

[5] M.P.J. Fromherz (2001), Constraint-based Scheduling, in *American Control Conference*, Arlington, VA.

[6] W.S. Gere (1966), Heuristics in Job Shop Scheduling, *Management Science* **13**, 167 – 190.

[7] S.K. Goyal, K. Mehta, R. Kodali, S.G. Deshmukh (1996), Simulation for Analysis of Scheduling Rules for a Flexible Manufacturing System, *Integrated Manufacturing Systems* **6**(5), 21 – 26.

[8] D. Montana (2005), A Comparison of Combinatorial Optimization and Dispatch Rules for Online Scheduling, *Proc. of the 2nd Multidisciplinary Conference on Scheduling: Theory and Application*, 353 – 362.

[9] S. Panwalker, W. Iskander (1977), A Survey of Scheduling Rules, *Operations Research* **25**(1), 45 – 61.

[10] thesis N. Policella (2005), Scheduling with Uncertainty - A Proactive Approach Using Partial Order Schedules, University of Rome "La Sapienza", Ph. D. Thesis.

[11] S.F. Smith (1994), OPIS: A Methodology and Architecture for Reactive Scheduling, 29 – 66, M. Zweben and M. Fox eds, *Intelligent Scheduling*, Morgan Kaufmann.

[12] R.-L. Sun, H.-X. Li, Y. Xiong (2006), Performance-Oriented Integrated Control of Production Scheduling, *IEEE Transactions on Systems, Man, and Cybernetics* **36**(4), 554 – 562.

[13] J.-P. Watson, J.C. Beck, A.A. Howe (2003), Problem Difficulty for Tabu Search in Job-Shop Scheduling, *Artificial Intelligence*.

Optimal and Approximate Periodic Task Scheduling with Storage Requirement Minimization

Sid-Ahmed-Ali Touati
University of Versailles, France. Sid.Touati@uvsq.fr

In this paper, we study an exact formulation of the general problem of one-dimensional periodic task scheduling under storage requirement, irrespective of machine constraints. We present a new theoretical framework that allows an early optimization of periodic storage requirement. This is based on inserting some storage dependence edges (*storage reuse* edges) labeled with *reuse distances* directly on the data dependence graph. In this new graph, we are able to fix the storage requirement measured as the exact number of necessary storage locations. The determination of storage and distance reuse is parametrized by the desired minimal execution period (resp. maximal execution throughput) as well as by the storage requirement constraints - either can be minimized while the other one is bounded, or alternatively, both are bounded.

Keywords: Multi-processor Scheduling.

1 Introduction

This article addresses the problem of storage optimization in cyclic data dependence graphs (DDGs), which is for instance applied in the practical problem of periodic register allocation for innermost loops on modern Instruction Level Parallelism (ILP) processors. The massive introduction of ILP processors since the last decade makes us re-think new ways of optimizing register/storage requirement in assembly codes before starting the instruction scheduling process under resource constraints. In such processors, instructions are executed in parallel thanks to the existence of multiple small computation units (adders, multipliers, load-store units, etc.). The exploitation of this new fine grain parallelism (at the assembly code level) asks to completely revisit the old classical problem of register allocation initially designed for sequential processors. Nowadays, register allocation has not only to minimize the storage requirement, but has also to take care of parallelism and total schedule time. In this research article, we do not assume any resource constraints (except storage requirement); Our aim is to analyze the trade-off between memory (register pressure) and parallelism in a periodic task scheduling problem. Note that this problem is abstract enough to be considered in other scheduling disciplines that worry about conjoint storage and time optimization in repetitive tasks.

Existing techniques in this field usually apply a periodic instruction scheduling that is sensitive to register/storage requirement. Therefore a great amount of work tries to schedule the instructions of a loop (under resource and time constraints) such that the resulting code does not use more than $\mathcal{R}$ values simultaneously alive. Usually they look for a schedule that minimizes the storage requirement under a fixed execution period [3, 4, 6]. In this paper, we satisfy register constraints early before instruction scheduling under resource constraints: we directly handle and modify the DDG in order to fix the storage requirement of any further subsequent periodic scheduling pass while taking care of not altering parallelism exploitation if possible. This idea uses the concept of reuse vector used for multi-dimensional scheduling [12, 13].

Our article is organized as follows. Sect. 2 defines our problem model. Sect. 3 starts the study with a motivating example. The problem of optimal periodic scheduling under storage constraints is described with graph theory and integer linear programming in Sect. 4. Sect. 5 presents simplified sub-problems and polynomial heuristics.

2 Tasks Model

We consider a simple innermost loop in a program represented by a set of l generic tasks (instructions) $T_0, \ldots, T_{l-1}$. Each task T_i should be executed n times, where n is the number of loop iterations. n is a an unknown, unbounded, but finite integer. This means that each task T_i has n instances. The k^{th} occurrence of task T_i is noted $T\langle i,k\rangle$, which corresponds to task executed at the k^{th} iteration of the loop, with $0 \leq k < n$.

The tasks (instructions) may be executed in parallel on an ILP processor. Each task may produce a result that is read/consumed by other tasks. The considered loop contains some data dependences represented with a graph $G = (V, E, \delta, \lambda)$ such that:

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{l-1}\}$.
- E is the set of edges representing precedence constraints (flow dependences or other serialization constraints). Any edge $e = (T_i, T_j) \in E$ has a latency $\delta(e) \in \mathbb{N}$ in terms of processor clock cycles and a distance $\lambda(e) \in \mathbb{N}$ in terms of number of loop iterations. The distance $\lambda(e)$ means that the edge $e = (T_i, T_j)$ is a dependence between the task $T\langle i,k\rangle$ and $T\langle j, k+\lambda(e)\rangle$ for any $k = 0, \ldots, n-1-\lambda(e)$.

We make a difference between tasks and precedence constraints depending whether they refer to data to be stored into registers or not

1. V_R is the set of tasks producing data to be stored into registers.
2. E_R is the set of flow dependence edges through registers. An edge $e = (T_i, T_j) \in E_R$ means that the task $T\langle i,k\rangle$ produces a result stored into a register and read/consumed by $T\langle j, k+\lambda(e)\rangle$. The set of consumers (readers) of a generic task T_i is then the set:

$$Cons(T_i) = \{T_j \in V \mid e = (T_i, T_j) \in E_R\}$$

Fig. 1 is an example of a data dependence graph (DDG) where bold circles represent V_R the set of generic tasks producing data to be stored into registers. Bold edges represent flow dependences (each sink of such edge reads/consumes the data produced by the source and stored in a register). Tasks that are not in bold circles are instructions that do not write into registers (write the data into memory or simply do not produce any data). Non-bold edges are other data or precedence constraints different from flow dependences. Every edge e in the DDG is labeled by the pair $(\delta(e), \lambda(e))$.

In our generic ILP processor model, we assume that the reading and writing from/into registers may be delayed from the starting time of task execution. Let assume $\sigma(T\langle i,k\rangle) \in \mathbb{N}$ as the starting execution time of task $T\langle i,k\rangle$. We thus define two delay functions δ_r and δ_w in which

$$\begin{array}{ll} \delta_w: & V_R \rightarrow \mathbb{N} \\ & T_i \mapsto \delta_w(T_i) |\ 0 \leq \delta_w(T_i) \\ & \text{the writing time of data produced by } T\langle i,k\rangle \text{ is } \sigma(T\langle i,k\rangle) + \delta_w(T_i) \\ \delta_r: & V \rightarrow \mathbb{N} \\ & T_i \mapsto \delta_r(T_i) |\ 0 \leq \delta_r(T_i) \\ & \text{the reading time of the data consumed by } T\langle i,k\rangle \text{ is } \sigma(T\langle i,k\rangle) + \delta_r(T_i) \end{array}$$

These two delays functions depend on the target processor and model almost all regular ILP hardware architectures (VLIW, EPIC/IA64 and superscalar processors). The next section recalls the definition of periodic task scheduling problem in case of one dimensional schedule.

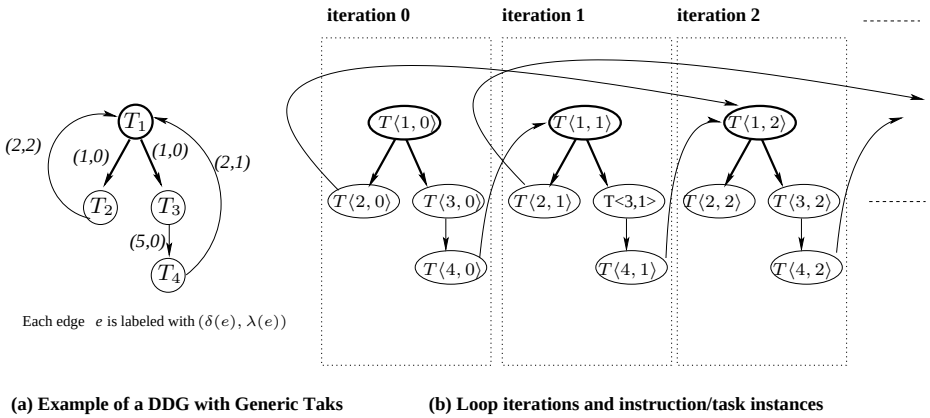


Figure 1: Example of Data Dependence Graphs with Recurrent Tasks

2.1 The Periodic Scheduling Problem

Instruction or task scheduling in our case is the process of assigning an integral execution date to each task occurrence. A schedule is considered as an integral function noted σ which must at least satisfy the precedence constraints defined by the DDG $G = (V, E, \delta, \lambda)$:

$$\forall e = (T_i, T_j) \in E, \forall k \in [0, n-1-\lambda(e)] : \ \sigma\big(T\langle i,k\rangle\big) + \delta(e) \leq \sigma\big(T\langle j, k+\lambda(e)\rangle\big) \tag{1}$$

However, since n the number of task occurrences is unknown and unbounded, we should not consider any shape of scheduling functions, even if they meet the constraints defined above. We should only look for *periodic* schedules since our aim is to generate a final compact code (a loop). A periodic scheduling function σ is associated with a unique integral period p (to be computed). The execution period p is integral and common to all generic tasks because it simplifies the code generation of the final loop. Other multi-dimensional periodic scheduling functions may be employed (with multiple periods, or with rational periods), but with the expense of huge code size. In our scope, a periodic scheduling function with a unique period p assigns to each generic task T_i an integral execution date for only the first task occurrence $T\langle i, 0\rangle$ that we note $\sigma_i = \sigma(T\langle i, 0\rangle)$. The execution date of any other occurrence $T\langle i, k\rangle$ becomes equal to $\sigma(T\langle i, k\rangle) = \sigma_i + k \times p$. By reporting this definition into Equ. 1, we get new periodic scheduling constraints that should be satisfied by σ:

$$\forall e = (T_i, T_j) \in E : \sigma_i + \delta(e) \leq \sigma_j + \lambda(e) \times p \tag{2}$$

Classically, by adding all such inequalities over any cycle C of the DDG G we find that p must be greater than or equal to $\max_C \left\lceil \frac{\sum_{e\in C}\delta(e)}{\sum_{e\in C}\lambda(e)} \right\rceil$, that we will denote in the sequel as the absolute Minimal Execution Period MEP. Computing MEP of a cyclic graph is a well known polynomial problem [2, 7]. The usual problem of periodic instruction scheduling looks for a schedule with a minimal period which satisfies additional constraints (resources, bounded storage requirement, etc.). In this article, we study the problem of periodic scheduling under data dependence and storage constraints.

2.2 Storage Requirement

When considering a periodic schedule σ with an integral period p, any generic task $T_i \in V_R$ corresponds to n task occurrences, each one producing a data at time $\sigma(T\langle i, k\rangle) + \delta_w(T_i)$, with $k = 0, \ldots, n-1$. Such data

should be stored inside a register until its last reading. Each flow dependence $e = (T_i, T_j) \in E_R$ means that the task occurrence $T\langle j, k + \lambda(e)\rangle$ reads the date produced by $T\langle i, k\rangle$ at time $\sigma_j + \delta_r(T_j) + (\lambda(e) + k) \times p$. The last reading time of a data produced by $T\langle i, k\rangle$ is called the *death* date and is equal to:

$$d_\sigma(T\langle i, k\rangle) = \max_{e=(T_i,T_j)\in E_R} \big(\sigma_j + \delta_r(T_j) + (\lambda(e) + k) \times p\big) \tag{3}$$

Every consumer that reads a data at its death time is called a *killer* of the data. Then, every data produced by $\sigma(T\langle i, k\rangle)$ is alive during a contiguous interval between the production date and the death date. It is called *lifetime interval* and is equal to:

$$LT_\sigma(T\langle i, k\rangle) =]\sigma(T\langle i, k\rangle) + \delta_w(T_i), d_\sigma(T\langle i, k\rangle)]$$

During this interval, the considered data is said *alive* and should reside inside a storage location (register) during the whole interval. The register assigned to store this data is not free to store another data during this lifetime interval. Let $alive^t_\sigma$ be the set of alive data at time $t \in \mathbb{N}$ when considering a schedule σ

$$alive^t_\sigma = \{T\langle i, k\rangle | T_i \in V_R, 0 \leq i < l,\ 0 \leq k < n,\ t \in LT_\sigma(T\langle i, k\rangle)\}$$

The storage requirement of the DDG G when considering the periodic schedule σ is equal to the maximal number of simultaneously alive data at any time. Formally, it is equal to:

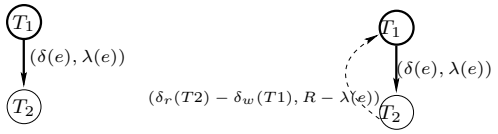
$$SR_\sigma(G) = \max_{t\in\mathbb{N}} \big|\{alive^t_\sigma\}\big| \tag{4}$$

When considering unbounded resources and unbounded storage/register facilities, we can easily compute an optimal periodic schedule, *i.e.*, with a minimal execution period $p = MEP$ [2]. We consider now a register pressure R (a finite number of available registers) and all the schedules that have a maximum of R simultaneously alive variables. Any register/storage allocation will induce new storage dependencies in the DDG; Hence register pressure has influence on the expected p even if we assume unbounded resources. What we want to analyze here is the minimum p that can be expected for any periodic schedule using at most R registers. We will denote this value as MEP_R and we will try to understand the relationship between MEP_R and R. Let us start by an example to fix the ideas.

3 Motivating Example

We give in this section more intuitions about the new edges that we add between two tasks in order to restrict the parallelism and hence restrict the whole storage requirement. These edges represent possible register reuse between tasks. This can be viewed as a variant of general storage mapping functions [12, 13].

Let us consider the DDG of Fig. 2(a). The DDG of this loop contains only one flow dependence $e = (T_1, T_2)$ with distance $\lambda(e) = 3$. If we have an unbounded number of registers, all iterations of this loop can be run in parallel since there is no cycle in the DDG. At each iteration $k = 0, \ldots, n-1$, the task $T\langle 1, k\rangle$ writes into a new register. Now, let us assume that we only have $R = 5$ available registers. The multiple instances of T_1 can use only $R = 5$ registers to periodically carry their data. In this case, the task $T\langle 1, k+R\rangle$ writes into the same register previously used by $T\langle 1, k\rangle$. This fact creates a *storage* dependence from $T\langle 2, k + \lambda(e)\rangle$ to $T\langle 1, k + R\rangle$; this is equivalent to inserting a storage dependence in the DDG from T_2 to T_1 with a distance $R - \lambda(e) = 2$. Since the writing of T_1 is delayed by $\delta_w(T_1)$ time units and the reading of T_2 is delayed by $\delta_r(T_2)$ time units, the latency of the introduced storage dependence is set to $\delta_r(T_2) - \delta_w(T_1)$. Consequently the DDG becomes cyclic because of storage limitations (see Fig. 2(b) where the storage dependence is dashed). The introduced dependence, also called *Universal Occupancy*



(a) Simple DDG (b) Storage Dependence

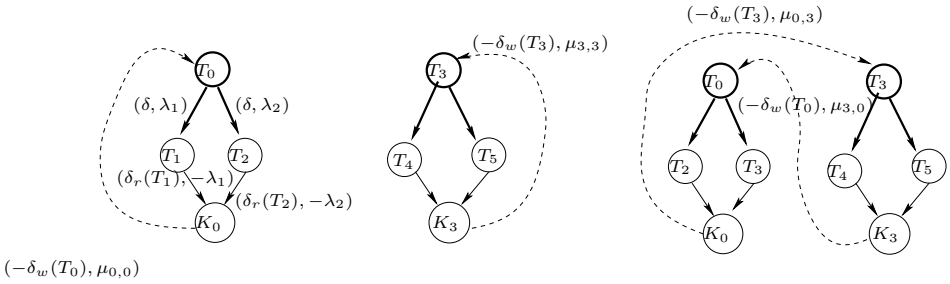
Figure 2: Simple Example

Vector [12], must in turn be counted when computing the new minimum execution period since a new cycle is created:

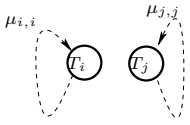
$$MEP_R \geq \frac{\delta(e) + \delta_r(T_2) - \delta_w(T_1)}{R}$$

When a task produces a data that is read by multiple consumers, and when the periodic schedule has not been fixed yet, we cannot know which consumer would be a killer of the data and hence we cannot know in advance when a register is freed. We propose a trick which defines for each task T_i a fictitious killing task K_i. We insert an edge from each consumer $T_j \in Cons(T_i)$ to K_i to reflect the fact that this killing task is scheduled after all the consumers of T_i (see Fig. 3). The latency of this edge is set to $\delta_r(T_j)$ because of the reading delay. We set its distance to $-\lambda$, where λ is the distance of the flow dependence between T_i and its consumer T_j. This is done to model the fact that the virtual task occurrence $K\langle i, k + \lambda - \lambda\rangle$, *i.e.* $K\langle i, k\rangle$, is scheduled after the death date of the data produced by $T\langle i, k\rangle$. The occurrence/iteration number k of the killer of $T\langle i, k\rangle$ is only a convention and can be changed by circuit retiming [8] without changing the fundamental mathematical problem.

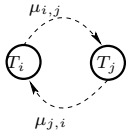
Now a register/storage allocation scheme consists of defining the edges and the distances of reuse. That is, we define for each T_i the task T_j and iteration distance $\mu_{i,j}$ such that $T\langle j, k + \mu_{i,j}\rangle$ reuses the same destination register as $T\langle i, k\rangle$. This reuse creates a new storage edge/dependence from K_i to T_j with latency $-\delta_w(T_j)$. The distance of this edge is $\mu_{i,j}$, to be defined/computed. We will see in a further section that the storage requirement can be expressed in terms of $\mu_{i,j}$.



(a) First Storage Reuse Decision (b) Another Storage Allocation Scheme



(c) First Reuse Graph



(d) Second Reuse Graph

Figure 3: Killing Tasks and Reuse Graphs

Hence controlling register pressure means first determining which instruction should reuse the register killed by another instruction (*where should storage dependences be added?*). Secondly we have to determine data lifetimes or equivalently storage requirement (*how many iterations later (μ) should reuse occur*)? As defined by the algebraic formulas of MEP, the lower is the μ the lower is the register requirement but also the larger is the MEP_R.

Fig. 3(a) presents a first reuse decision where each generic task reuses the register freed by itself. This is illustrated by adding a storage dependence from K_0 (resp. K_3) to T_0 (resp. T_3) with an appropriate distance μ as we will see later. Another reuse decision (see Fig. 3(b)) may be that the generic task T_0 (resp. T_3) reuses the register freed by T_3 (resp. T_0). This is illustrated by adding a storage dependence from K_0 (resp. K_3) to T_3 (resp. T_0). In both cases, the register pressure is equal to the sum of all μ distances, but it is easy to see that the two schemes do not have the same impact on MEP_R: intuitively, it is better that the operations share registers instead of using two different pools of registers. For this simple example with two tasks, we have only two choices for reuse decisions. However a general loop with l tasks producing data, we may have an exponential number of possible reuse decisions.

There are three main constraints that the resulting DDG must meet. First it must be schedulable by periodic scheduling. Second, the number of registers used by any storage allocation scheme must be lower or equal to the number of available registers. Third and last, the critical ratio (MEP_R) must be kept as lower as possible in order to save task parallelism. The next section gives a formal definition of the problem and provides an exact formulation.

4 Exact Problem Formulation with Graph Theory and Integer Programing

The storage reuse relationship between the generic tasks is described by defining a new graph called *a reuse graph*. Fig. 3(c) shows the first reuse decision where T_i (resp. T_j) reuses the register used by itself $\mu_{i,i}$ ($\mu_{j,j}$ resp.) iterations earlier. Fig. 3(d) is the second reuse choice where T_i (resp. T_j) reuses the register used by T_j (resp. T_i) $\mu_{j,i}$ (resp. $\mu_{i,j}$) iterations earlier. The resulting DDG after adding the killing tasks and the storage dependences to apply the register reuse decisions is called the *DDG associated with a reuse graph*: Fig. 3(a) is the associated DDG with Fig. 3(c), and Fig. 3(b) is the one associated with Fig. 3(d). In the next section, we give a formal definition and model of the storage allocation problem based on reuse graphs. We denote by $G_{\rightarrow r}$ the DDG associated with the reuse graph.

4.1 Reuse Graphs

A storage allocation consists of choosing which task reuses which released register. We define:

Definition 1 (Reuse Graph) *Let $G = (V, E, \delta, \lambda)$ be a DDG. The reuse graph $G^r = (V_R, E_r, \mu)$ is defined by the set of tasks V_R, the set of edges representing storage reuse relationship, and edge distances μ. Two tasks are connected in G^r by an edge $e = (T_i, T_j)$ iff $T\langle j, k + \mu(e)\rangle$ reuses the register freed by $T\langle i, k\rangle$.*

We call E_r the set of reuse edges and $\mu(e)$ a reuse distance. Given $G^r = (V_R, E_r, \mu)$ a reuse graph, we report the storage reuse relationship to the DDG $G = (V, E, \delta, \lambda)$ by adding a storage dependence from K_i to T_j iff $e = (T_i, T_j)$ is a reuse edge. The distance of this dependence is $\mu(e)$. The introduction of these extra edges into G produces the DDG $G_{\rightarrow r}$ associated with the reuse graph G^r.

A reuse graph must obey some constraints to be valid:

1. The resulting associated DDG $G_{\rightarrow r}$ must be schedulable, *i.e.*, it has at least one periodic schedule;
2. Each task must reuse only one freed register, and each register must be reused by only one task.

Lemma 1 *[11] Let $G^r = (V_R, E_r, \mu)$ be a valid reuse graph for a DDG $G = (V, E, \delta, \lambda)$. Then:*

- *the reuse graph consists of only elementary and disjoined cycles;*
- *any task $T_i \in V_R$ belongs to a unique cycle in the reuse graph.*

First, let us assume a reuse graph for a DDG G. If such reuse graph is valid, we can build a periodic storage allocation for G, as explained in the following theorem. We require $\mu(G^r)$ registers, in which $\mu(G^r)$ is the sum of all μ distances in the reuse graph G^r.

Theorem 1 *[11] Let $G = (V, E, \delta, \lambda)$ be a DDG and $G^r = (V_R, E_r, \mu)$ be a valid reuse graph. Then the reuse graph defines a periodic storage allocation with $\mu(G^r)$ registers. Formally:*

$$\forall \sigma \text{ a periodic schedule for } G_{\rightarrow r} : \ SR_\sigma(G_{\rightarrow r}) \leq \mu(G^r)$$

From all above, we deduce a formal definition of the problem of optimal periodic storage allocation with maximal execution throughput. We call it Periodic Scheduling with Storage Minimization (PSSM).

Problem 1 (PSSM) *Let $G = (V, E, \delta, \lambda)$ be a data dependence graph and p a desired execution period. Find a valid reuse graph such that the associated DDG $G_{\rightarrow r}$ is schedulable with a a period equal to p while $\mu(G^r)$ is minimal.*

This problem is NP-complete [11]. In practice, the problem of register allocation is sightly different. The processor has $\mathcal{R}$ a finite number of registers and we should find a time-optimal schedule such that the storage requirement is below the limit $\mathcal{R}$. In this case, the problem can be re-stated as : Find a valid reuse graph such that $R = \mu(G^r) \leq \mathcal{R}$ while the associated DDG $G_{\rightarrow r}$ is schedulable with a period equal to $p = MEP_R$. It is straightforward to see that PSSM can be iteratively used to solve the problem of register allocation. The following section gives an integer linear formulation for the PSSM problem.

4.2 Exact Formulation for PSSM

In this section, we give an integer linear model for solving PSSM. It is built for a fixed desired period p. Our PSSM exact model uses the linear formulation of the logical implication ($\Longrightarrow$) by introducing binary variables [11].

Basic Variables

- a schedule variable $\sigma_i \geq 0$ for each task $T_i \in V$, including σ_{K_i} for each killing node K_i. We assume a finite upper bound L for such schedule variables (L sufficiently large, $L = \sigma_{e \in E} \delta(e)$).
- a binary variables $\theta_{i,j}$ for each $(T_i, T_j) \in V_R^2$. It is set to 1 iff (T_i, T_j) is a reuse edge;
- $\mu_{i,j}$ reuse distances for all $(T_i, T_j) \in V_R^2$.

Linear Constraints

- Data dependences (see Equ. 2):$\forall e = (T_i, T_j) \in E : \ \sigma_i - \sigma_j \leq -\delta(e) + p \times \lambda(e)$
- Killing dates for consumed data:
  $\forall T_i \in V_R$, $\forall T_j \in Cons(T_i) | e = (T_i, T_j) \in E_R$: $\sigma_{K_i} \geq \sigma_j + \delta_r(T_j) + p \times \lambda(e)$
- There is a storage dependence between k_i and T_j if (T_i, T_j) is a reuse edge:

$$\forall (T_i, T_j) \in V_R^2 : \theta_{i,j} = 1 \Longrightarrow \sigma_{K_i} - \delta_w(T_j) \leq \sigma_j + p \times \mu_{i,j}$$

- If there is no register reuse between two tasks T_i and T_j, then $\theta_{i,j} = 0$. The storage dependence distance $\mu_{i,j}$ must be set to 0 in order to not be accumulated in the objective function.

$$\forall (T_i, T_j) \in V_R^2 : \theta_{i,j} = 0 \Longrightarrow \mu_{i,j} = 0$$

The reuse relation must be a bijection from V_R to V_R:

- a register can be reused by one task: $\forall T_i \in V_R : \sum_{T_j \in V_R} \theta_{i,j} = 1$
- a task can reuse one released register: $\forall T_i \in V_R : \sum_{T_j \in V_R} \theta_{j,i} = 1$

Objective Function We want to minimize the storage requirement: Minimize $\sum_{(T_i,T_j) \in V_R^2} \mu_{i,j}$.

If we want to consider the real problem of periodic register allocation, it is sufficient to find a solution below $\mathcal{R}$ the number of available registers, *i.e.*, we do not need to minimize the storage requirement at the lowest possible level. In this case, we can avoid defining an objective function, and we can simply add a constraint: $\sum_{(T_i,T_j) \in V_R^2} \mu_{i,j} \leq \mathcal{R}$

5 Problem Simplification : Fixing Reuse Edges

Buffers [1, 5, 9] are FIFO storage facilities that transport data between repetitive tasks. That is, a generic task writes its result in its own buffer that will be accessed (read) by further tasks. When buffer/FIFO structures are used as a storage memory, there is no storage sharing between generic tasks, since each task has its own pool of storage locations. In our framework, this problem actually amounts to deciding that each generic task reuses the same storage location, possibly some iterations later. Therefore we consider now the complexity of our storage minimization problem when fixing reuse edges. This generalizes the buffer minimization approach. Formally, the problem can be stated as follows.

Problem 2 (Fixed PSSM) *Let $G^r = (V_R, E_r, \mu)$ an incomplete reuse graph with already fixed reuse edges, but the reuse distances remain to be computed. Let $G_{\rightarrow r} = (V, E, \delta, \lambda)$ an incomplete DDG associated with it, and p a desired execution period. Let $E' \subseteq E$ be the set of already fixed storage dependences (which correspond to the reuse edges of G^r). Find a distance $\mu_{i,j}$ for each storage dependence $(K_i, T_j) \in E'$ such that $\sum_{(K_i,T_j) \in E'} \mu_{i,j}$ is minimal, and the resulted DDG $G_{\rightarrow r}$ has a valid schedule with a period equal to p.*

In the following, we assume that $E' \subseteq E$ is the set of these already fixed storage dependences (their distances have to be computed). The process of early fixing storage dependences greatly simplifies the integer linear program of Sect. 4.2. Consequently, the Fixed PSSM problem can be solved by the following integer program, assuming a given desired execution period p. Let first define the integer variables used in our exact formulation:

- a schedule variable $\sigma_i \geq 0$ for each task $T_i \in V$, including σ_{K_i} for each killing node K_i. We assume a finite upper bound L for such schedule variables (L sufficiently large, $L = \sigma_{e \in E} \delta(e)$).
- $\mu_{i,j}$ reuse distances for all fixed storage dependences $(K_i, T_j) \in E'$.

The following integer program solves the Fixed PSSM problem:

$$\begin{array}{ll} \text{Minimize} & \sum_{(K_i,T_j) \in E'} \mu_{i,j} \\ \text{Subject to:} & \\ p \times \mu_{i,j} + \sigma_j - \sigma_{K_i} \geq -\delta_w(T_j) & \forall (K_i, T_j) \in E' \\ \sigma_j - \sigma_i \geq \delta(e) - p \times \lambda(e) & \forall e = (T_i, T_j) \in E - E' \\ \sigma_{K_i} - \sigma_j \geq \delta_r(T_j) + p \times \lambda(e) & \forall T_i \in V_R, \forall T_j \in Cons(T_i) | e = (T_i, T_j) \in E_R \end{array} \quad (5)$$

If we want to solve the dual problem, *i.e.*, to compute MEP_R the minimal execution period given a storage capacity R, it is sufficient to find a solution below R the number of available storage elements. In this case, we do not need to minimize the storage requirement at the lowest possible level, we can simply remove the objective function and we add a constraint:$\sum_{(K_i,T_j)\in E'} \mu_{i,j} \leq R$. MEP_R can be calculated iteratively using a binary search on p between MEP and the upper-bound L.

System 5 contains $\mathcal{O}(|V|)$ variables and $\mathcal{O}(|E|))$ linear constraints, but it does not mean that its resolving complexity is polynomial. Indeed, even if System 5 is a good simplification of PSSM, its constraints matrix isn't totally unimodular yet. However, this simplification allows to solve the PSSM for larger DDGs (multiple hundreds of nodes) compared to the case of exact optimal PSSM where only small DDGs can be taken into account (since optimal PSSM is an NP-complete problem).

Furthermore, we can go further by simplifying System 5. Since p is a constant, we can do the variable substitution $\mu' = p \times \mu$ and System 5 becomes:

$$\begin{array}{ll} \text{Minimize} & \sum_{(K_i,T_j)\in E'} \mu'_{i,j} \\ \text{Subject to:} & \\ \mu'_{i,j} + \sigma_j - \sigma_{K_i} \geq -\delta_w(T_j) & \forall (K_i,T_j) \in E' \\ \sigma_j - \sigma_i \geq \delta(e) - p \times \lambda(e) & \forall e = (T_i,T_j) \in E - E' \\ \sigma_{K_i} - \sigma_j \geq \delta_r(T_j) + p \times \lambda(e) & \forall T_i \in V_R, \forall T_j \in Cons(T_i) | e = (T_i,T_j) \in E_R \end{array} \quad (6)$$

Theorem 2 *[11] The constraints matrix of the integer linear program of System 6 is totally unimodular, i.e., the determinant of each square sub-matrix is equal to 0 or to* $\pm$ *1.*

Consequently, we can use polynomial algorithms to solve this problem [10]. This would allow us to consider huge DDGs (multiple thousands of nodes). We must be aware that the back substitution $\mu = \frac{\mu'}{p}$ may produce a non integral value for the distance μ. If we ceil it by setting $\mu = \lceil \frac{\mu'}{p} \rceil$, a sub-optimal solution may result[1].

It is easy to see that the loss in terms of number of storage requirement is not greater than the number of generic tasks that write into a storage location ($|V_R|$).

6 Conclusion

This article presents a new theoretical approach for periodic task scheduling with storage requirement optimization. Our theoretical framework is more generic than the existing ones, since it allows exact definition of the problem while considering delays in writing and reading from storage locations. As a practical application, we use it for solving the problem of periodic register allocation for instruction scheduling on ILP processors, which is a distinct problem and more difficult than the old classical register allocation for sequential processors.

Our exact formulation of the problem has many applications : we can fix an execution period while minimizing the storage requirement, or we can fix the execution period while bounding the storage requirement. The dual problem of bounding the storage requirement while minimizing the execution period can be solved by a binary search on p (solving iteratively successive integer problems of PSSM).

Storage allocation is expressed in terms of reuse edges and reuse distances to model the fact that two tasks use the same storage location. Our integer linear program computes an optimal solution with reduced constraint matrix size, and enables us to make a tradeoff between task parallelism and number of storage locations.

Since computing an optimal periodic storage allocation is intractable in large data dependence graphs, we identified one approximate subproblem by fixing reuse edges and computing the reuse distances that minimize the overall storage requirement. We can use it in different ways, as setting self-reuse edges (buffers

[1] Of course, if we have $p = 1$ (case of non cyclic DDG for instance), the solution remains equal to System 5.

minimization) or fixing arbitrary (or with a cleverer algorithm) other reuse cycles. This simplification allow us to consider DDGs with multiple hundreds of nodes, but do not define a polynomial instance of the original problem. A polynomial subproblem is obtained thanks to a non-bijective variable substitution $\mu' = p \times \mu$. This simplification allows us to consider huge DDGs, but induce an additional cost in terms of storage requirement (sub-optimal result).

References

[1] A. Munier Kordon and J.-B. Note (2005), A Buffer Minimization Problem for the Design of Embedded Systems, *European Journal of Operational Research* **164**(3), 669 – 679.

[2] C. Hanen and A. Munier (1995), A Study of the Cyclic Scheduling Problem on Parallel Processors, *Discrete Applied Mathematics* **57**(2-3), 167 – 192.

[3] A. E. Eichenberger, E. S. Davidson, and S. G. Abraham (1996), Minimizing Register Requirements of a Modulo Schedule via Optimum Stage Scheduling, *International Journal of Parallel Programming* **24**(2), 103 – 132.

[4] D. Fimmel and J. Muller (2001), Optimal Software Pipelining Under Resource Constraints, *International Journal of Foundations of Computer Science (IJFCS)* **12**(6), 697 – 718.

[5] J. Korst (1992), *Periodic Multiprocessor Scheduling*. PhD thesis, Eindhoven University of Technology. The Netherlands.

[6] J. Janssen (2001), *Compilers Strategies for Transport Triggered Architectures*, PhD thesis, Delft University, Netherlands.

[7] E.L. Lawler (1972), Optimal Cycles on Graphs and Minimal Cost-to-Time Ratio Problem. In A. Marzojlo, editor, *Periodic Optimization*, volume 1, 38 – 58, Springer-Verlag.

[8] C.E. Leiserson and J.B. Saxe (1991), Retiming Synchronous Circuitry, *Algorithmica* **6**, 5 – 35.

[9] Q. Ning and G.R. Gao (1993), A Novel Framework of Register Allocation for Software Pipelining, In *Conference Record of the Twentieth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 29 – 42, Charleston, South Carolina, ACM Press.

[10] A. Schrijver (1987) *Theory of Linear and Integer Programming*, John Wiley and Sons, New York.

[11] S.-A.-A. Touati (2002), *Register Pressure in Instruction Level Parallelisme*, PhD thesis, Université de Versailles, France, ftp.inria.fr/INRIA/Projects/a3/touati/thesis.

[12] M. M. Strout, L. Carter, J. Ferrante, and B. Simon (1998), Schedule-Independent Storage Mapping for Loops, *ACM SIG-PLAN Notices* **33**(11), 24 – 33.

[13] W. Thies, F. Vivien, J. Sheldon, and S. Amarasinghe (2001), A Unified Framework for Schedule and Storage Optimization, *ACM SIGPLAN Notices* **36**(5), 232 – 242.

Large-Scale Short-Term Planning in Chemical Batch Production

Norbert Trautmann
Department of Business Administration, University of Bern, 3012 Bern, Switzerland,
norbert.trautmann@ifm.unibe.ch

Christoph Schwindt
Institute of Management and Economics, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany,
christoph.schwindt@tu-clausthal.de

In the chemical industry, final products arise from chemical and physical transformations of materials in processing units. We consider the case of batch production mode, where the total requirements for intermediate and final products are divided into individual batches. To produce a batch, the inputs are first loaded into a processing unit, then a transformation process is executed, and finally the output is unloaded from the processing unit. In general, storage facilities of limited capacity are available for stocking raw materials, intermediates, and final products.

We present a novel cyclic approach to solving large-scale instances of the minimum-makespan production scheduling problem. This problem can be decomposed into a batching and a batch scheduling problem. The basic idea of the cyclic approach consists in reducing the size of the batch scheduling problem by computing a cyclic sub-schedule, which needs to be executed several times. Using a mixed-integer nonlinear programming formulation of the batching problem one can compute the set of batches of one cycle and the number of cycles needed to satisfy the primary requirements. The sub-schedule is then obtained by scheduling the batches on the processing units subject to material-availability and storage-capacity constraints. In an experimental performance analysis, we applied this cyclic approach to a set of 70 test instances. For each instance, we obtained a better feasible solution within much less CPU time than a state-of-the-art method from the literature.

Keywords: Applications, Production Scheduling, Process Scheduling, Large Scale Scheduling

1 Planning problem

Short-term planning of batch production in the chemical industry deals with the detailed allocation of the production resources of a single plant over time to the processing of given primary requirements for final products. Batch production is typically used either for technological reasons or for the case of multiple products processed on multi-purpose equipment. In Subsection 1.1 we review the particular characteristics of batch production on multi-product production plants. In Subsection 1.2 we state the planning problem. In Subsection 1.3 we introduce a practical example of a chemical production plant that has been provided by Kallrath (2002).

1.1 Batch production

In general, a multi-product plant consists of multi-purpose processing units (e.g., heaters, filters, and reactors) and storage facilities (e.g., tanks, silos, and a cooling house). The final products are produced by performing a sequence of transformations, which are also called tasks. For executing a task, several alternative processing units may be available. In this case, the duration of the task may depend on the processing unit used. In a multi-purpose processing unit, several processes

can be performed, but only one at a time. Between consecutive executions of different tasks in a processing unit, a cleaning with sequence-dependent duration may be necessary.

Each task consumes and produces one or several products, where the input or output proportions are either fixed or variable within prescribed bounds. Some intermediates are perishable and must be consumed immediately after production. Material flows can be linear, divergent, convergent, or general (including the case of recycling flows).

The minimum and maximum filling levels of the processing unit used give rise to a lower and an upper bound on the batch size. This is the reason for executing a task several times to fulfill the primary requirements. Note that in chemical batch production, the duration of a task is independent of the batch size. In the following, the execution of a task will be called an operation.

Each product family requires a specific configuration of the plant. During a re-configuration of the plant, no operation can be processed. Thus, the objective of makespan minimization is particularly important in order to ensure high resource utilization and short customer lead times.

1.2 Short-term planning problem

The planning problem can be stated as follows. Given primary requirements for the final products, we must determine (a) the batch size, the input and the output proportions, and the number of executions for each task; (b) an assignment of the corresponding operations to the processing units; and (c) start times of the operations such that

- the given primary requirements for final products are satisfied,
- the prescribed intervals for the batch sizes and the input and output proportions are observed,
- no processing unit processes more than one operation at a time,
- the processing units are cleaned between consecutive operations,
- a sufficient amount of each input product is available at the start of each operation,
- sufficient storage space for output products is available at the completion of each operation,
- all perishable intermediates are consumed immediately after production, and
- the makespan is minimized.

1.3 Sample production process

In this subsection we describe the chemical batch production process of the case study presented by Kallrath (2002) and based on an existing plant. For the representation we use a modification of the state-task network (STN) concept introduced by Kondili et al. (1993). An STN is a directed graph which includes three types of elements:

1. *State nodes* represent the raw materials, intermediates, and final products. They are drawn as ellipses labeled with the respective state number and the initial and maximum stocks of the corresponding product. Some of the intermediate products cannot be stocked, which is indicated by the label "ns" (no stock). The value ∞ for the initial or maximum stock means that there is respectively sufficient initial stock or storage capacity.
2. *Task nodes* refer to the chemical or physical transformations of materials from one or more input states into one or more output states. Task nodes are represented by rectangles indicating the task number, the processing units in which the task can be executed, and the corresponding processing and cleaning times.

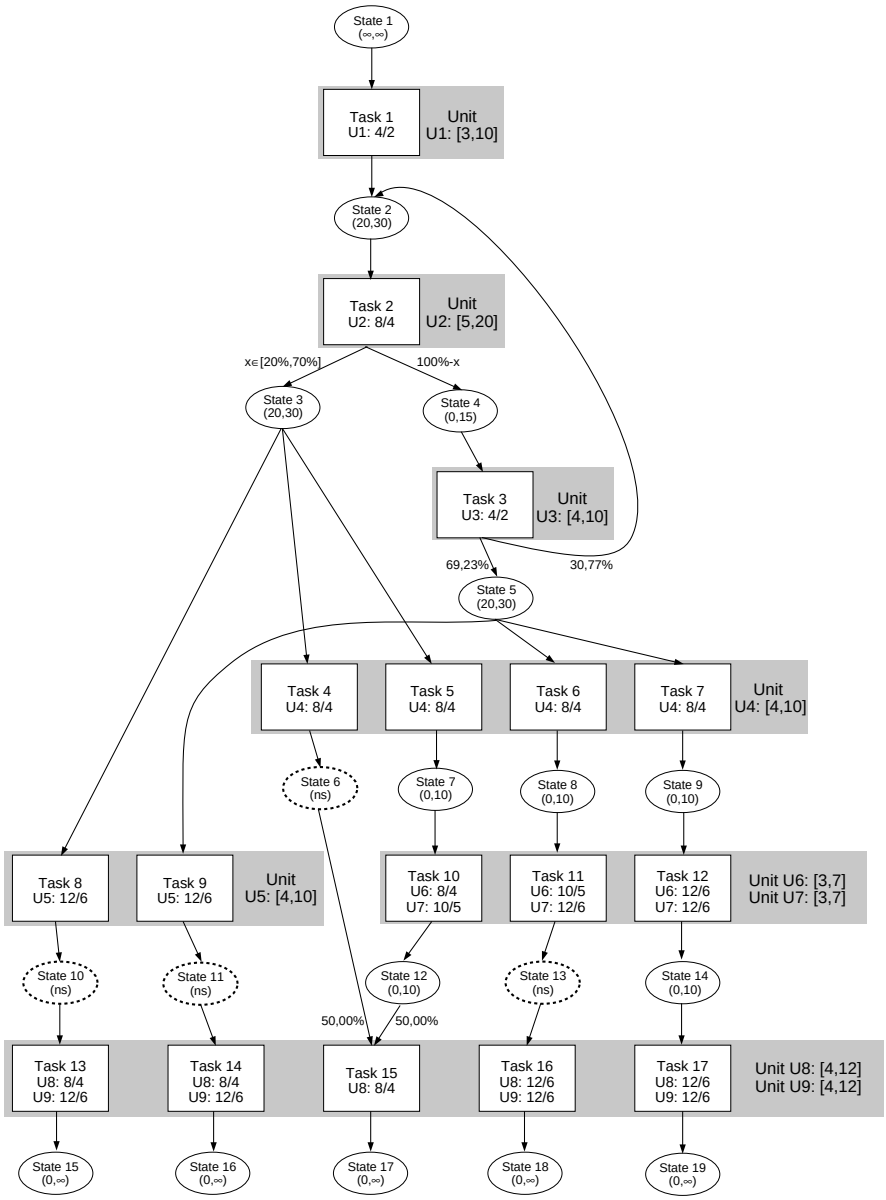


Figure 1: State-task network of the chemical batch production process

3. *Arcs* indicate the flow of material. If more than one input product is consumed or more than one output product is produced, the possible values of the input or output proportions are shown on the arcs.

Figure 1 shows the STN for the batch production process under study with 19 products, 17 tasks, and 9 processing units. The shaded areas group the tasks that can be processed in the same units. Alternative processing units are available for executing tasks 10 to 14, 16, and 17. In order to guarantee product purity, each processing unit must be cleaned before proceeding to an operation at a higher task index (tasks are numbered according to increasing quality requirements). The time needed for cleaning a processing unit equals one half of the processing time of the preceding operation.

2 Related literature

In the context of supply chain management, the primary requirements to be produced in the production network are determined on the mid-term campaign planning level. Campaign planning aims at using the procurement, production, storage, and transportation facilities in the supply chain efficiently by synchronizing the respective material flows. For mixed-integer linear programming models for campaign planning, we refer to Grunow et al. (2002) and Timpe and Kallrath (2000).

The short-term planning problem described in Subsection 1.2 has been widely discussed in the chemical engineering literature. An overview of state-of-the-art models and methods can be found in the survey papers of Floudas and Lin (2004), Burkard and Hatzl (2005), and Méndez et al. (2006). Roughly speaking, monolithic approaches (cf. Subsection 2.1) and decomposition approaches (cf. Subsection 2.2) can be distinguished.

2.1 Monolithic approaches

The monolithic solution approaches address the short-term planning problem as a whole, starting from a mixed-integer linear programming formulation. The time horizon is divided into a given number of time periods. In so-called time-indexed formulations (see e.g., Kondili et al. 1993), the period length is fixed. In contrast, in so-called continuous-time formulations (see e.g., Ierapetritou and Floudas 1998 or Castro et al. 2001), the period length is chosen implicitly during the solution of the mixed-integer linear program.

The main disadvantage of all these monolithic approaches is that the CPU time required for solving real-world problems tends to be prohibitively long (cf. Maravelias and Grossmann 2004). To overcome this difficulty, Shah et al. (1993), Blömer and Günther (2000), and others have developed different heuristics that aim at reducing the number of variables. Nevertheless, the computational burden for solving real-world problems with more than 50 operations is still very high.

2.2 Decomposition approaches

Promising alternative approaches are based on decomposing the short-term planning problem into interdependent subproblems. Decomposition methods have for example been proposed by Brucker and Hurink (2000), Neumann et al. (2002), and Maravelias and Grossmann (2004).

Brucker and Hurink (2000) did not consider all the constraints mentioned in Subsection 1.1. In particular, they assumed that the capacity of the storage facilities is unlimited, and that each task can be executed in one dedicated processing unit. The authors have devised a constructive algorithm for computing the numbers and the sizes of the batches, and a tabu search procedure for scheduling the batches on the processing units.

Maravelias and Grossmann (2004) proposed computing the number of batches by solving the LP relaxation of a monolithic continuous-time formulation of the short-term planning problem. The batch sizes and the start times of the operations were then determined by a branch-and-bound algorithm that uses constraint-propagation techniques.

The solution approach proposed in the present paper is based on a hierarchical decomposition into a batching and a batch-scheduling problem presented in Neumann et al. (2002). The solution of the batching problem provides the numbers and the sizes of all batches for the intermediate and final products needed to satisfy the primary requirements. The batch scheduling problem consists in allocating the processing units, intermediates, and storage facilities over time to the processing of the operations arising from the batching step. In Neumann et al. (2002), the batching problem was formulated as a mixed-integer nonlinear program which is of moderate size and can be solved using standard mathematical programming software. Neumann et al. (2002) and Schwindt

and Trautmann (2004) have developed a truncated branch-and-bound method and a priority-rule-based method, respectively, for solving the batch scheduling problem. Within a reasonable amount of computation time, good feasible solutions to problem instances with up to 100 operations can be computed with both methods. Gentner et al. (2004) proposed a decomposition of the batch scheduling problem which partitions the set of all batches into a sequence of subsets. The assignment of the batches to the individual subsets is determined stepwise by solving a binary linear program in each iteration. Gentner et al. (2004) and Gentner (2005) computed feasible solutions to batch scheduling instances with up to 3000 operations. However, for such large-scale instances, this method requires several hours of CPU time.

3 Cyclic solution approach

In this section we present a cyclic approach to the short-term planning problem, which is based on the decomposition principle introduced by Neumann et al. (2002). A preliminary version of our approach can be found in Schwindt and Trautmann (2006).

Our method consists of the three phases of cyclic batching (Subsection 3.1), cyclic batch scheduling (Subsection 3.2), and concatenation (Subsection 3.3). Each of these phases is performed only once. Moreover, we limit the size of the cyclic batch scheduling problem to be solved. In total, a relatively short CPU time is required, and we are able to efficiently cope with problem instances including thousands of operations.

3.1 Cyclic batching

In the cyclic batching phase, we determine the set of operations (together with their respective batch sizes and input and output proportions) belonging to one cycle, and the number of cycles needed to satisfy the given primary requirements. In doing so we must take into account the prescribed bounds on the batch sizes, the prescribed bounds on the input and output proportions, and the initial inventory levels. Moreover, in order to keep the scheduling problem tractable, we impose an upper bound on the total number of operations belonging to one cycle. To obtain a cyclic solution allowing for executing the same sub-schedule an arbitrary number of times, the amount of any intermediate produced within one cycle must be equal to the amount consumed. This cyclic batching problem can be formulated as a mixed-integer nonlinear program of moderate size (cf. Schwindt and Trautmann 2006), and locally optimal solutions can be determined using standard software.

3.2 Cyclic batch scheduling

In the cyclic batch scheduling phase, we compute a sub-schedule by allocating the processing units and storage facilities over time to the processing of the operations belonging to one cycle such that the makespan is minimized. An appropriate sub-schedule can be determined using the truncated branch-and-bound method or the priority-rule-based method proposed by Neumann et al. (2002) and Schwindt and Trautmann (2004), respectively. An improved version of the latter procedure is presented in Fink and Schwindt (2007).

The main principle of the priority-rule-based method consists in scheduling the operations one after another on the processing units in such a way that the material-availability constraints are observed. The storage-capacity constraints are taken into account in a second scheduling pass (Schwindt and Trautmann's method) or by appropriately delaying producing operations via an unscheduling procedure (Fink and Schwindt's method).

3.3 Concatenation

In the concatenation step, we generate a complete production schedule as follows. The computed sub-schedule for executing the operations of one cycle defines a partial ordering among those operations. We represent this ordering by precedence relationships between the operations. Moreover, the completion time of the last operation that is processed in a processing unit defines a release date for the changeover to the first operation in that unit in the next execution of the sub-schedule. Analogously, the last change in the inventory level of an intermediate gives rise to a release date for the first operation that subsequently produces or consumes that intermediate.

The start and completion times for the operations in the first cycle equal those of the sub-schedule computed in the cyclic batch scheduling phase. For computing the start and completion times of the operations in the next cycle, we solve a temporal scheduling problem, which consists in computing an earliest schedule for those operations subject to the precedence relationships between and the release dates for the operations. This temporal scheduling problem represents a longest path problem and can be solved efficiently by standard network flow algorithms (see Ahuja et al., 1993). Thus, the concatenation of the cyclic sub-schedules forming the complete production schedule can be performed in polynomial time.

4 Performance analysis

We compared our cyclic approach to the decomposition method devised by Gentner et al. (2004). For our analysis, we used a test set proposed by Gentner (2005), which consists of 70 instances generated by varying the primary requirements for the final products in the example presented in Subsection 1.3. For each instance we computed an approximate solution to the cyclic batching problem using Frontline Systems' Solver package. The sub-schedules were generated with a randomized multi-pass version of Schwindt and Fink's priority-rule-based method (2007). We performed the tests on an 3.4 GHz Pentium IV PC. The results for the method of Gentner et al. have been reported in Gentner (2005) and refer to a 1.4 GHz Pentium IV PC.

The results obtained for the 70 problem instances are shown in Table 1, where "$C_{\max}$" stands for the best makespan found, "t_{cpu}" is the CPU time in seconds, and "# op.'s" designates the number of operations in the complete production schedule. For each problem instance the new method was able to find a markedly better solution. Especially for large-scale problem instances, the required CPU time was significantly smaller than the time needed by the method of Gentner et al. Having prescribed an upper bound of $\overline{\varepsilon} = 150$ batches, between 14 and 583 seconds were required for solving the cyclic batching problem. The priority-rule based method was stopped after 60 seconds of CPU time. The concatenation required less than one second of CPU time.

5 Conclusions

In this paper we have presented a cyclic approach to short-term planning in chemical batch production. Our method is based on the decomposition of the short-term planning problem into a batching level providing the set of operations to be executed and a batch scheduling level that schedules the operations on the processing units subject to material-availability and storage-capacity constraints. The main idea of our cyclic approach consists in formulating the batching problem as a cyclic model where the given primary requirements are produced through the repetitive execution of the same set of operations. In this way we ensure that the resulting batch scheduling problem can be solved within a reasonable amount of computation time by computing a subschedule for the operations of one cycle and concatenating the number of cycles needed to meet the primary requirements.

Table 1: Computational results

	Gentner (2005)		This paper		
Instance	$C_{\max}$	t_{cpu}	# op.'s	$C_{\max}$	t_{cpu}
WeKa0_0	178	18	88	128	116
WeKa0_1	352	38	176	252	113
WeKa0_2	474	53	264	376	118
WeKa0_3	612	120	352	500	119
WeKa0_4	738	209	440	624	115
WeKa0_5	906	178	528	748	122
WeKa0_6	1046	215	616	872	119
WeKa0_7	1199	323	704	996	121
WeKa0_8	1334	281	792	1120	117
WeKa0_9	1548	399	880	1244	128
WeKa0_10	1740	431	968	1368	100
WeKa0_15	2123	644	1408	1988	97
WeKa0_20	2899	1500	1848	2608	97
WeKa0_30	4416	5235	2728	3884	77
WeKa19_0	238	19	105	166	80
WeKa19_1	436	165	210	316	81
WeKa19_2	618	59	315	466	79
WeKa19_3	818	97	420	616	80
WeKa19_4	1004	179	525	766	81
WeKa19_5	1184	232	630	916	80
WeKa19_6	1384	330	735	1066	83
WeKa19_7	1570	474	840	1216	81
WeKa19_8	1806	442	945	1366	81
WeKa19_9	1946	568	1050	1516	80
WeKa19_10	2135	570	1155	1666	83
WeKa19_15	2848	1322	1680	2416	79
WeKa19_20	3811	1911	2205	3166	78
WeKa19_30	5896	6610	3255	4666	76
WeKa20_0	168	34	89	138	86
WeKa20_1	336	50	178	264	87
WeKa20_2	590	72	267	390	90
WeKa20_3	750	76	356	516	100
WeKa20_4	896	93	445	642	98
WeKa20_5	990	126	534	768	100
WeKa20_6	1138	184	623	894	95
WeKa20_7	1294	215	712	1020	95
WeKa20_8	1547	200	801	1146	96
WeKa20_9	1816	327	890	1272	94
WeKa20_10	1920	448	979	1398	94
WeKa20_15	2386	421	1424	2028	96
WeKa20_20	3604	969	1869	2658	96
WeKa20_30	5194	3255	2759	3918	75
WeKa21_0	210	17	98	144	103
WeKa21_1	382	127	196	284	100
WeKa21_2	555	67	294	424	95
WeKa21_3	728	97	392	564	91
WeKa21_4	868	152	490	704	86
WeKa21_5	1082	226	588	844	86
WeKa21_6	1224	250	686	984	83
WeKa21_7	1420	240	784	1124	82
WeKa21_8	1554	291	882	1264	85
WeKa21_9	1701	475	980	1404	85
WeKa21_10	1916	469	1078	1544	82
WeKa21_15	2545	771	1568	2244	81
WeKa21_20	3398	1415	2058	2944	82
WeKa21_30	5091	5957	3038	4344	89
WeKa22_0	190	192	102	152	327
WeKa22_1	376	85	204	290	644
WeKa22_2	558	102	306	428	298
WeKa22_3	722	120	408	566	155
WeKa22_4	930	249	510	704	239
WeKa22_5	1024	239	612	842	324
WeKa22_6	1298	255	714	980	270
WeKa22_7	1488	341	816	1118	150
WeKa22_8	1520	439	918	1256	276
WeKa22_9	1779	427	1020	1394	149
WeKa22_10	1786	647	1122	1532	221
WeKa22_15	2586	704	1632	2222	171
WeKa22_20	3172	1598	2142	2912	206
WeKa22_30	5375	7563	3162	4292	271

The performance of the new method has been tested on a test set involving instances with several thousands of operations.

An important area of our future research will be the adaptation of the cyclic approach to continuous process scheduling problems where tasks are executed at constant production rates. In addition, we are developing predictive-reactive methods for the short-term planning of chemical production plants when processing times, resource availabilities, or production yields are subject to uncertainty. The different short-term planning methods will be integrated with decision models for mid-term multi-site campaign planning in the chemical industry.

Acknowledgements. This research has been supported by the Deutsche Forschungsgemeinschaft under Grant Schw1178/1.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin (1993), *Network Flows*, Prentice Hall, Englewood Cliffs.

[2] F. Blömer, H.O. Günther (2000), LP-based heuristics for scheduling chemical batch processes. *Int. J. Prod. Res.* **38**, 1029 – 1051.

[3] P. Brucker, J. Hurink (2000), Solving a chemical batch scheduling problem by local search, *Annals of Oper. Res.* **96**, 17 – 36.

[4] R.E. Burkard, J. Hatzl (2005), Review, extensions and computational comparison of MILP formulations for scheduling of batch processes, *Comp. Chem. Eng.* **29**, 1752 – 1769.

[5] P. Castro, A.P. Barbosa-Póvoa, H. Matos (2001), An improved RTN continuous-time formulation for the short-term scheduling of multipurpose batch plants, *Ind. Eng. Chem. Res.* **40**, 2059 – 2068.

[6] R. Fink, C. Schwindt (2007), A priority-rule based method for predictive-reactive batch production scheduling in the process industries. *Report PLC–3*, Institute of Management and Economics, Clausthal University of Technology.

[7] C.A. Floudas, X. Lin (2004), Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review, *Comp. Chem. Eng.* **28**, 2109 – 2129.

[8] K. Gentner (2005), *Dekompositionsverfahren für die ressourcenbeschränkte Projektplanung*, Shaker, Aachen.

[9] K. Gentner, K. Neumann, C. Schwindt, N. Trautmann (2004), Batch production scheduling in the process industries. In: J.Y.T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, Chapter 48, 1 – 21.

[10] M. Grunow, H.O. Günther, M. Lehmann (2002), Campaign planning for multi-stage batch processes in the chemical industry, *OR Spectrum* **24**, 281 – 314.

[11] M.G. Ierapetritou, C.A. Floudas (1998), Effective continuous-time formulation for short-term scheduling: 1. Multipurpose batch processes, *Ind. Eng. Chem. Res.* **37**, 4341 – 4359.

[12] J. Kallrath (2002), Planning and scheduling in the process industry, *OR Spectrum* **24**, 219 – 250.

[13] E. Kondili, C.C. Pantelides, R.W.H. Sargent (1993), A general algorithm for short-term scheduling of batch operations — I. MILP Formulation, *Comp. Chem. Eng.* **17**, 211 – 227.

[14] C.T. Maravelias, I.E. Grossmann (2004), A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants, *Comp. Chem. Eng.* **28**, 1921 – 1949.

[15] C.A. Méndez, J. Cerdá, I.E. Grossmann, I. Harjunkoski, M. Fahl (2006), State-of-the-art review of optimization methods for short-term scheduling of batch processes, *Comp. Chem. Eng.* **30**, 913 – 946.

[16] K. Neumann, C. Schwindt, N. Trautmann (2002), Advanced production scheduling for batch plants in process industries, *OR Spectrum* **24**, 251 – 279.

[17] C. Schwindt, N. Trautmann (2004), A priority-rule based method for batch production scheduling in the process industries. In: D. Ahr, R. Fahrion, M. Oswald, G. Reinelt (eds.), *Operations Research Proceedings 2003*, Springer, Berlin, 111 – 118.

[18] C. Schwindt, N. Trautmann (2006), A cyclic approach to large-scale short-term planning of multi-purpose batch plants. In: M. Morlock, C. Schwindt, N. Trautmann, J. Zimmermann (eds.), *Perspectives on Operations Research*, Deutscher Universitäts-Verlag, Wiesbaden, 225 – 238.

[19] N. Shah, C.C. Pantelides, R.W.H. Sargent (1993), A general algorithm for short-term scheduling of batch operations — II. Computational Issues, *Comp. Chem. Eng.* **17**, 229 – 244.

[20] C.H. Timpe, J. Kallrath (2000), Optimal planning in large multi-site production networks, *Eur. J. Oper. Res.* **126**, 422 – 435.

Single Machine and Parallel Machine Scheduling Problems with a Common Due Date to Minimize Total Weighted Tardiness

Nguyen Huynh Tuong, Ameur Soukhal, Jean-Charles Billaut
Laboratoire d'Informatique, Université François Rabelais Tours, France, nguyen.huynh@etu.univ-tours.fr,
{ameur.soukhal,jean.billaut}@univ-tours.fr

This paper deals with a common due date parallel machines scheduling problem in which each job has a different tardiness penalty. The objective is to minimize the total weighted tardiness. The scheduling problem of minimizing the total weighted tardiness with a common due date on a single machine is known to be ordinary NP-hard. This is also the case for problem $Pm|d_i = d|\sum w_iT_i$. A new dynamic programming algorithm is proposed to solve the $1|d_i = d|\sum w_iT_i$ scheduling problem and a fully polynomial time approximation scheme (FPTAS) is deduced from this algorithm. We show that the complexity of this FPTAS is better than existing one. These results are generalized to the parallel machines scheduling case.

Keywords: Multi-processor Scheduling; Weighted Tardiness; Dynamic Programming Algorithm; Approximation Scheme.

1 Introduction

Lawler and Moore [5] are the first ones who paid attention to tardiness scheduling problems for the last 40 years. In this paper, we consider the scheduling problem where n jobs $J_1, \ldots, J_n$ have to be scheduled without preemption on m identical parallel machines ($m \geq 1$). Each job is described by a processing time p_i, a positive integer tardiness penalty w_i, and a common due date $d_i = d, d \geq 0$. All jobs are available at time zero. We denote by S_i the starting time of J_i and by C_i its completion time.

As defined in [2], a job J_i can belong to one of the following three states:

1. early job if $C_i < d$,
2. fully-tardy job if $S_i > d$,
3. straddling job if $S_i < d$ and $C_i \geq d$.

The tardiness of J_i, denoted by T_i is defined by $T_i = max(0, C_i - d)$. So, if J_i is completed before the due date ($C_i < d$) there is no penalty. Otherwise, there is a job-dependent tardiness penalty given by $w_i \times T_i$. The objective is to minimize the sum of tardiness penalties ($\sum_{i=1}^{n} w_iT_i$). According to the standard scheduling notation, the considered problem is denoted by $Pm|d_i = d|\sum w_iT_i$, $m \geq 1$.

If $m = 1$, the problem $1|d_i = d|\sum w_iT_i$ is proved to be NP-hard in the ordinary sense [8]. In [5], the authors propose a dynamic programming algorithm with complexity $O(n^2d)$. Kolliopoulos and Steiner [3] determine a fully polynomial-time approximation scheme (FPTAS) for the problem $1|d_i = d|\sum w_i(T_i + d)$. So, even an optimal sequence for $1|d_i = d|\sum w_i(T_i + d)$ is also optimal in the case of minimizing $\sum w_iT_i$ but an ϵ-approximation for the first cannot guarantee the same approximation scheme for the second.

For problem $1|d_i = d|\sum w_iT_i$, Kellerer and Strusewich [2] determine a FPTAS with complexity $O((n^6 logW)/\epsilon^3)$ (W is the sum of weights).

In the case of m identical parallel machines, the problem $Pm|d_i = d|\sum w_iT_i$, for a fixed m ($m \geq 2$), is NP-Hard and does not accept a polynomial ρ-approximation algorithms. In fact, the special case $Pm|d_i = d|\sum T_i$ where $w_i = 1$, $(\forall i, i = 1, \ldots, n)$, is NP-Hard [1] and does not accept any polynomial ϵ-approximation algorithms with $\epsilon < \infty$ unless $P = NP$ [4]. However, in [4] the authors propose two approximation algorithms with guaranted performances. They show that the value of the solution given by one of the two approximation algorithms, denoted by X^0, satisfies the following inequality $(X^0 - X^*)/(X^* + d) \leq \epsilon$ (X^* corresponds to optimal solution value).

In this study, we propose both dynamic programming algorithms and FPTAS for the single machine and parallel machine scheduling problem. However, the FPTAS for m parallel machines is valid under condition $P > md$ where P is the total processing time.

This paper is organized as follows. Section 2 is dedicated to the single machine scheduling problem ($m = 1$). In this section, we describe the new proposed dynamic programming algorithm with complexity $O(n^2d)$. Then, we will show how to transform this algorithm to a FPTAS, which can be runned in $O(h(W, P, d) \times n^3/\epsilon)$ time where W is the sum of weights, and $h(W, P, d)$ is a polynomial function of W, P and d. In Section 3, the DP algorithm and the approximation algorithm are extended to the parallel machine scheduling problem.

2 Single machine scheduling problem

This section deals with problem $1|d_i = d|\sum w_iT_i$. We remark that if $d = 0$, the problem is equivalent to $1||\sum w_iC_i$ that can be solved in $O(n \log n)$. In the case where $d \neq 0$ we propose an optimal pseudo-polynomial time algorithm with a complexity $O(n^2d)$.

The two following properties are well known [5] or trivial:

Property 1 : There always exist an optimal schedule where:

- there is no idle times between jobs,
- the early jobs are scheduled in an arbitrary order,
- the fully-tardy jobs are scheduled in a non-decreasing order of p_i/w_i.

Property 2: There is an optimal solution where the early jobs are scheduled according to a non-decreasing order of p_i/w_i.

To find an optimal solution we have to determine the straddling job and the set of early jobs. Finding the early job set can be viewed as solving a knapsack problem, where the objective is to maximize the penalty cost. So, we propose a dynamic programming algorithm where an optimal solution is calculated according to the following three steps.

1. at iteration k, let J_k be the straddling job.
2. we consider an initial solution where all jobs except J_k are scheduled after date d, i.e. there is no early job. To minimize the sum of weighted tardiness penalty of this initial solution, the jobs are scheduled according to the non-decreasing order of p_i/w_i, called WSPT.
3. we determine an optimal set of early jobs of total length strictly less than d.

It means that we have n choices for the straddling job. To illustrate the DP algorithm, let us study the following example. At the kth iteration, the job J_k is considered as the straddling job.

Let J_{i_1}, J_{i_2} and J_{i_3} be the only three jobs that can be early (the processing time of the other jobs exceeds d) (see Fig. 1). Let $Z^{(k)}(i)$ be the cost of job J_i in the initial solution and $F^{(k)}(i)$ be the reduction cost that corresponds to moving J_i to the early set. Let $Z^{(k)}$ be the total cost that we have to maximize at iteration k. The final cost is denoted by $F^{(k)}(n)$.

In the initial solution, let $n1$ be the number of jobs scheduled between J_k and J_{i_1}, $n2$ be the number of jobs between J_{i_1} and J_{i_2}, $n3$ be the number of jobs between J_{i_2} and J_{i_3}, $n4$ be the number of jobs after J_{i_3}. We have $n = n1 + n2 + n3 + n4 + 4$.

Let $W_{n1}^{(k)}$ be the sum of the weights of the $n1$ jobs, $W_{n2}^{(k)}$ be the sum of the weights of the $n2$ jobs, $W_{n3}^{(k)}$ be the sum of the weights of the $n3$ jobs, etc.

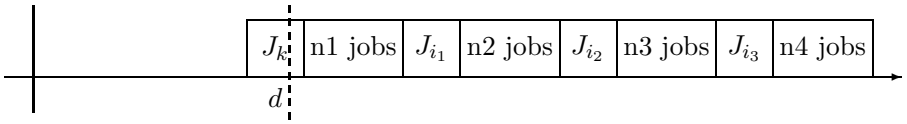


Figure 1: example - first step

First, if J_{i_1} is early, it means that this job is scheduled without tardy penalty (see Fig. 2) and the reduction cost is given by the following formula:

$$F^{(k)}(J_{i_1}) = Z^{(k)}(J_{i_1}) + p_{J_{i_1}} \times (W_{n2}^{(k)} + w_{J_{i_2}} + W_{n3}^{(k)} + w_{J_{i_3}} + W_{n4}^{(k)})$$

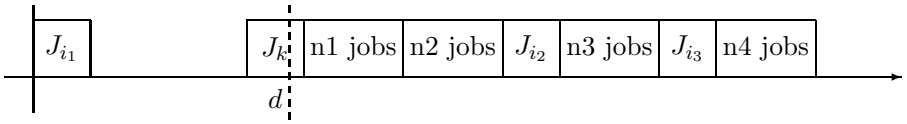


Figure 2: example - second step

By moving J_{i_2} in the early job set (see Fig. 3), we have:

$$F^{(k)}(J_{i_2}) = F^{(k)}(J_{i_1}) + Z^{(k)}(J_{i_2}) + p_{J_{i_2}}.(W_{n3}^{(k)} + w_{J_{i_3}} + W_{n4}^{(k)}) - p_{J_{i_1}}.w_{J_{i_2}}$$

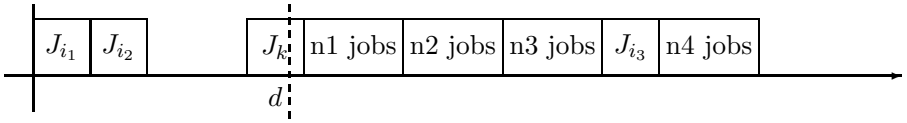


Figure 3: example - third step

Finally, $F^{(k)}(J_{i_3}) = F^{(k)}(J_{i_2}) + Z^{(k)}(J_{i_3}) + p_{J_{i_3}} W_{n4}^{(k)} - w_{J_{i_3}}(p_{J_{i_1}} + p_{J_{i_2}})$ (see Fig. 4).

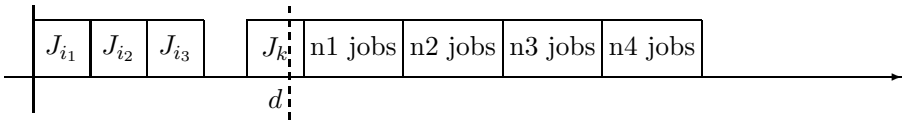


Figure 4: example - last step

To solve the problem, we propose the following algorithm, noted WTSMCDD (**W**eighted **T**ardiness scheduling problem in a **S**ingle **M**achine with **C**ommon **D**ue **D**ate).

For each $k, k = 1, \ldots, n$, we have $(n-1) \times d$ iterations. Consequently, the algorithm complexity is $O(n^2 d)$. This algorithm gives a global optimal solution since we can determine a local optimal solution for each sub-problem corresponding to the straddling job k. Of course, at each iteration k and at any time $0 \leq t < d$, $F^{(k)}(n,t)$ defines a sub-set of early jobs, sequenced according to the WSPT rule. The reduction given by this sub-set corresponds to the large reduction of the total cost of solution k. So, we can compare all the possible cases for $S_k = t$ where $d - p_k \leq S_k < d$. This comparison can be done according to the following formula:

$$Z^{(k)} - F^{(k)}(n,t) + (S_k + p_k - d)w_k + (S_k + p_k - d)W'^{(k)}(n,t) = Z^{(k)} - F^{(k)}(n,t) + (t + p_k - d)(W'^{(k)}(n,t) + w_k)$$

where $Z^{(k)} - F^{(k)}(n,t)$ is the weighted tardiness of the late job; $(S_k + p_k - d)w_k$ is the weighted tardiness of the straddling job J_k, and $(S_k + p_k - d)(W'^{(k)}(n,t)$ is the weighted tardiness defined by the late job set's movement at the time $(S_k + p_k - d)$.

Let's remind that in Lawler's solution, the early jobs are scheduled according to WLPT rule (**W**eighted **L**ongest **P**rocessing **T**ime). However, in our solution the early jobs are scheduled according to the WSPT rule. This DP algorithm allows finding a FPTAS.

In the following section, we propose a FPTAS based on the dynamic programming algorithm. In [2], the authors propose an FPTAS with complexity $O((n^6 \log W)/\epsilon^3)$. Even the complexity of our DP algorithm have the same one to the DP of Lawler and Moore [5], the FPTAS we propose has a complexity in $O(Wn^3/\epsilon)$.

Algorithm 1 WTSMCDD: $1|d_i = d| \sum w_i T_i$

```
1:  // The jobs are supposed to be sorted according to WSPT //
2:  for k = 1 to n do
3:      // Let J_k be the straddling job //
4:      Z^(k)(i) = w_i T_i; Z^(k) = sum_{(i=1...n, i≠k)} Z^(k)(i)
5:      F^(k)(0,0) = 0 ; F^(k)(0,t) = -∞, if t ≠ 0
6:      W^(k)(0) = sum_{j=1}^{n}(w_j) - w_k ; W'^(k)(0,t) = W^(k)(0)
7:      for i = 1 to n, i ≠ k do
8:          Z^(k)(i) = w_i T_i; W^(k)(i) = W^(k)(i-1) - w_i
9:          for t = 0 to d - 1 do
10:             F^(k)(i,t) = max(F^(k)(i-1,t), F^(k)(i-1,t-p_i) + Z^(k)(i) + p_i W^(k)(i) - w_i(t-p_i))
11:             if (F^(k)(i,t) = F^(k)(i-1,t)) then
12:                 W'^(k)(i,t) = W'^(k)(i-1,t)
13:             else
14:                 W'^(k)(i,t) = W'^(k)(i-1,t-p_i) - w_i
15:             endif
16:         endfor
17:     endfor
18:     S^(k) = min_{t=d-p_k}^{d-1}(Z^(k) - F^(k)(n,t) + (t + p_k - d)(W'^(k)(n,t) + w_k)
19: endfor
20: The optimal schedule corresponds to the sequence minimizing S^(k).
```

Figure 5: Algorithm WTSMCDD

2.1 Lower bound

To prove that the considered problem accepts a FPTAS, we first calculate a lower bound. Let LB be the lower bound. We have $LB = w_{min} \times (P - d)$, where $w_{min} = \min_{i=1,\dots,n}\{w_i\}$ and $P = \sum_{i=1,\dots,n} p_i$ (the total processing time).

It is easy to show that LB is a lower bound. Of course, if $P \leq d$ we have $LB \leq 0 = \sum(w_i T_i)$. Otherwise, there exists at least one late job (a straddling job). The last job in a given sequence must be penalized by $w_{[n]}(P - d) \geq w_{min}(P - d)$. This penalty is less than or equal to the total weighted tardiness of the sequence.

2.2 Approximation algorithm

For a feasible schedule π, let $T_i(\pi)$ ($C_i(\pi)$, $S_i(\pi)$) be the tardiness of job J_i (the completion time of job J_i, and the starting time of job J_i) in sequence π; let $T(\pi)$ ($X(\pi)$) be the total tardiness cost (the total weighted tardiness cost) of sequence π.
Following the idea presented in [7] to determine a FPTAS for knapsack problem, we define a new instance as follows:

- Given $\epsilon > 0$, let $W = \sum_{i=1}^{n}(w_i)$ and

$$K = \frac{\epsilon LB}{Wn} = \frac{\epsilon w_{min}(P - d)}{Wn} \tag{1}$$

- For each job J_i, $i = 1, \dots, n$, we define a new processing time as follows: $p'_i = \lfloor \frac{p_i}{K} \rfloor$.
- Let $d' = \frac{d}{K}$.
- Let $X'(\pi_A)$ be the total weighted tardiness cost that corresponds to the optimal solution π_A given by the previous algorithm WTSMCDD applied on the new processing times p'_i.

Let π^* be the optimal schedule and let $X(\pi^*)$ be its value. We have

$$X'(\pi_A) \leq X'(\pi^*) \tag{2}$$

and

$$X(\pi_A) \geq X(\pi^*) \tag{3}$$

For any job J_i, we have:

$$Kp'_i \leq p_i \tag{4}$$

and

$$p_i \leq K(p'_i + 1) \tag{5}$$

¿From (4), we have: $\Rightarrow K\sum_{j=1}^{n}(p'_j) \leq \sum_{j=1}^{n}(p_j)$
Therefore, since there is no iddle time between jobs, for the sequence π, we have:
$\Rightarrow KC'_i(\pi) \leq C_i(\pi)$
$\Rightarrow KC'_i(\pi) - d \leq C_i(\pi) - d$
$\Rightarrow KT'_i(\pi) \leq T_i(\pi)$
$\Rightarrow Kw_iT'_i(\pi) \leq w_iT_i(\pi)$
$\Rightarrow KX'_i(\pi) \leq X_i(\pi)$ (6)
From (5) : $\Rightarrow \sum_{j=1}^{n}(p_j) \leq K\sum_{j=1}^{n}(p'_j + 1)$
$\Rightarrow \sum_{j=1}^{n}(p_j) \leq Kn + K\sum_{j=1}^{n}(p'_j)$

$\Rightarrow C_i(\pi) \leq Kn + KC'_i(\pi)$
$\Rightarrow C_i(\pi) - d \leq Kn + KC'_i(\pi) - Kd'$
$\Rightarrow T_i(\pi) \leq Kn + KT'_i(\pi)$
$\Rightarrow w_iT_i(\pi) \leq w_iKn + w_iKT'_i(\pi)$
$\Rightarrow X(\pi) \leq WKn + KX'(\pi)$
Precisely, for the sequence π_A we have: $\Rightarrow X(\pi_A) \leq KX'(\pi_A) + WKn$
From (2) $\Rightarrow X(\pi_A) \leq KX'(\pi^*) + WKn$
From (6) $\Rightarrow X(\pi_A) \leq X(\pi^*) + WKn$
From (1) $\Rightarrow X(\pi_A) \leq X(\pi^*) + \epsilon w_{min}(P - d)$
$\Rightarrow X(\pi_A) \leq X(\pi^*) + \epsilon LB$
$\Rightarrow X(\pi_A) \leq (1 + \epsilon)X(\pi^*)$

Thus, the running time of the algorithm is $O(n^2d') = O(n^2d/K) = O(\frac{dW}{w_{min}(P-d)}.\frac{n^3}{\epsilon})$ which is polynomial in n and $(1/\epsilon)$. So, we have a fully polynomial time approximation scheme.

Remark: If $d \leq \frac{Pw_{min}}{w_{min}+1}$, the complexity becomes more simple and corresponds to $O(\frac{Wn^3}{\epsilon})$.

3 Identical parallel machines scheduling problem

In this section, the scheduling problem $Pm|d_i = d|\sum w_iT_i$ with fixed m is studied. It is easy to show that there exists an optimal solution with at least one job scheduled on each machine ($n \geq m$). Consequently, we have the following property:

Property 3 : There exists an optimal solution with exactly m straddling jobs.

Proof: Let π^* be an optimal sequence. In π^* we suppose that there is not straddling job on a machine M_j. Let J_k be the last job from the early job set scheduled on M_j in π^* (i.e. $C_k < d$). Then we can move the job J_k to the right until $C_k = d$ without increasing the tardiness penalty. By definition J_k becomes the jth straddling job.

So, it is possible to construct an optimal solution with idle times between early job set and the straddling job without increasing the total tardiness penalties.

Let's consider the case $m = 2$. As in the case of a single machine, we assume that the jobs are sorted according to WSPT rule. Firstly we choose two straddling jobs and the remaining jobs are executed on the first machine after the completion time of the straddling job assigned to the first machine. It means that all jobs are late. Then, we try to assign each job to one machine in order to be scheduled in the time interval $[0, d]$. If it is not possible, the current job is assigned to the second machine and scheduled after the completion date of the straddling job in the second machine. So the complexity of this algorithm is $O(n^3d^2p^2_{max}P)$ where $p_{max} = \max^n_{p_i=1}\{p_i\}$ and $P = \sum^n_{p_i=1}\{p_i\}$ (see Algorithm 6).

Hence, following the same idea and for any fixed $m \geq 2$ we can determine a DP algorithm with a complexity $O(n^{m+1}d^mp^m_{max}P^{m-1})$, which is pseudo-polynomial.

In [4] the authors show that there is no polynomial ϵ-approximation algorithm for problem $Pm|d_i = d|\sum^n_{i=1} T_i$ with $\epsilon < \infty$ unless $P = NP$.

However, if $P > md$ then we can show that the considered problem accepts a FPTAS. It is obtained by following the same idea of the FPTAS proposed in the case of a single machine scheduling problem. However, K is given by: $K = \frac{\epsilon LB_m}{Wn}$, where LB_m is a lower bound given by $LB_m = w_{min}Z$ where Z corresponds to the optimal value of $Pm||C_{max}$.

Algorithm 2 WTP2CDD: $P2|d_i = d|\sum w_i T_i$

Order the jobs according to increasing ratios p_i/w_i
for $k \in \{1 \dots n\}$ **and** $l \in \{1 \dots n\}$ **and** $k \neq l$ **do**
  /*Let J_k, J_l be the straddling jobs in the first and second machine, respectively.*/
  for $S_k \in \{d - p_k, \dots, d-1\}$ **and** $S_l \in \{d - p_l, \dots, d-1\}$ **do**
  Calculate the tardiness cost $T^{(k,l,S_k,S_l)}(i)_{\ i \notin l,k}$ such that J_i begins after C_k
  $Z^{(k,l,S_k,S_l)}(i) = w_i T^{(k,l,S_k,S_l)}(i)$
  $Z^{(k,l,S_k,S_l)} = \sum_{(i=1,i\neq k,l)}^{n} Z^{(k,l,S_k,S_l)}(i)$
  $F^{(k,l,S_k,S_l)}(0,0,0,S_l + p_l - d) = 0$
  $F^{(k,l,S_k,S_l)}(0,t_1,t_2,t_3) =$ -∞, if $t_1 \neq 0$, $t_2 \neq 0$ and $t_3 \neq (S_l + p_l - d)$
  $W^{(k,l,S_k,S_l)}(0) = \sum_{j=1}^{n}(w_j) - w_k - w_l$
  $W'^{(k,l,S_k,S_l)}(0,t) = W^{(k,l,S_k,S_l)}(0)$
  for $i \in \{1, \dots, n\}, i \neq k, i \neq l$ **do**
    $Z^{(k,l,S_k,S_l)}(i) = w_i T_i$
    $W^{(k,l,S_k,S_l)}(i) = W^{(k,l,S_k,S_l)}(i-1) - w_i$
    for $t_1 \in \{0, \dots, S_k\}$ **and** $t_2 \in \{0, \dots, S_l\}$ **and** $t_3 \in \{0, \dots, P\}$ **do**
      $m_0 = F^{(k,l,S_k,S_l)}(i-1,t_1,t_2,t_3)$
      $m_1 = F^{(k,l,S_k,S_l)}(i-1,t_1 - p_i,t_2,t_3) + Z^{(k,l,S_k,S_l)}(i) + p_i W^{(k,l)}(i) - w_i(t_1 + t_2 + t_3 - p_i)$
      $m_2 = F^{(k,l,S_k,S_l)}(i-1,t_1,t_2 - p_i,t_3) + Z^{(k,l,S_k,S_l)}(i) + p_i W^{(k,l)}(i) - w_i(t_1 + t_2 + t_3 - p_i)$
      $m_3 = F^{(k,l,S_k,S_l)}(i-1,t_1,t_2,t_3 - p_i) + Z^{(k,l,S_k,S_l)}(i) + p_i W^{(k,l)}(i) - w_i(t_1 + t_2 + t_3 - p_i) - w_i t_3$
      $F^{(k,l,S_k,S_l)}(i,t_1,t_2,t_3) = \max\{m_0, m_1, m_2, m_3\}$
    endfor
  endfor
  $S^{(k,l)}(S_k,S_l) = \min_{t_1=0}^{S_k} \min_{t_2=0}^{S_l} \min_{t_3=0}^{P} \left(Z^{(k,l,S_k,S_l)} - F^{(k,l,S_k,S_l)}(n,t_1,t_2,t_3) \right)$
  $S(k,l) = \min_{S_k = d-p_k \dots (d-1), S_l = d-p_l \dots (d-1)} S^{(k,l)}(S_k,S_l)$
  endfor
endfor
The optimal schedule corresponds to the sequence defined by the straddling jobs J_k and J_l in which S is minimized.

Figure 6: Algorithm WTP2CDD

So, $LB_m \leq w_{min}(P - md) > 0$ since $P > md$. Hence, the FPTAS's complexity is bounded by $O(H\frac{n^{3m}}{\epsilon^{2m-1}})$ with $H = d^m p_{max}^m P^{m-1}(\frac{W}{w_{min}(P-md)})^{2m-1}$.

In the case where $d = 0$, the DP algorithm can solve the $Pm||\sum w_i C_i$ problem.

4 Conclusion

This paper deals with single machine and identical parallel machines scheduling problems with a common due date. The objective is to minimize the total weighted tardiness penalties. A new

DP algorithm is developped for the single machine scheduling problem with a complexity identical to the complexity of Lawler's DP. The DP allows defining a FPTAS with complexity less than the best one of the literature. We also show how to transform these results in order to solve the case with m identical parallel machines.

References

[1] M.R. Garey, D.J. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

[2] H. Kellerer, Vitaly A. Strusevich (2006), A fully polynomial approximation scheme for the single machine weighted total tardiness problem with a common due date, *Theoretical Computer Science* **369**, 230 – 238.

[3] S.G. Kolliopoulos, G.Steiner (2005), Approximation algorithms for scheduling problems with a modified total weighted tardiness objective, *Computing and Software Technical Reports 2004-2005*.

[4] M. Y. Kovalyov, Frank Werner (2002), Approximation Schemes for Scheduling Jobs with Common Due Date on Parallel Machines to Minimize Total Tardiness, *Journal of Heuristics* **8**, 415 – 428.

[5] E.L. Lawler, J.M. Moore (1969), A functional equation and its application to resource allocation and sequencing problems, *Management Science* **16**, 77 – 84.

[6] J.K. Lenstra, A.H.G. Rinnooy Kan (1978), Complexity of scheduling under precedence constraints, *Operations Research* **26**, 22 – 35.

[7] V. V. Vazirani (2003), *Approximation Algorithms*, Springer-Verlag, Berlin, Heidelberg.

[8] J. Yuan (1992), The NP-hardness of the single machine common due date weighted tardiness problem, *Systems Science and Mathematical Sciences* **5**, 328 – 333.

A Combined Meta-Heuristic with Hyper-Heuristic Approach to the Scheduling of the Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Machines

José Antonio Vázquez Rodríguez, Sanja Petrovic
School of Computer Science and Information Technology, University of Nottingham Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, U.K., {jav, sxp}@cs.nott.ac.uk

Abdellah Salhi
Department of Mathematical Sciences, the University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, U.K., as@essex.ac.uk

This paper is concerned with makespan minimisation in a hybrid flow shop with sequence dependent setup times and uniform parallel machines. A lower bound on the optimum makespan is derived and several variants of the genetic algorithm with a hybrid representation are proposed. This representation consists of a permutation, that are commonly used in meta-heuristic approaches to flow shop problems, and a list of dispatching rules to be called upon to complete a schedule. The latter resembles what has been recently termed hyper-heuristics (heuristics that "manage" other heuristics). The proposed methods are used to solve instances generated with real world data from a company. Encouraging results are reported.

Keywords: Production Scheduling, Evolutionary Algorithms, Heuristic Search, Hyper-Heuristics.

1 Introduction

This work is motivated by the scheduling problem of a real world Hybrid Flow Shop (HFS) with Sequence Dependent Setup Times (SDST) and Uniform Parallel Machines (UPM), (HFS-SDST-UPM) encountered in a cardboard box shop. The objective function to consider is the maximum completion time of operations, i.e. the makespan. This problem belongs to the class of difficult problems. Even simplified variants of it are NP-hard, [1]. The HFS is encountered in many forms (each in turn considering different constraints and objective functions) in applications such as ceramic tiles manufacturing, [2], web service architectures, [3], and others.

In [4], the authors proposed a Genetic Algorithm (GA) which combines a meta-heuristic with a hyper-heuristic for an HFS problem. Hyper-heuristic is the term which refers to multi-level heuristics in which a high level heuristic coordinates lower level ones [5]. In the chromosome representation of this GA, a permutation represents a schedule for the first stage of the shop and a sequence of dispatching rules for the rest (one for each of the remaining stages). This representation was tested against many other GA representations and compared favourably. However, the proposed meta-hyper-heuristic was not tested against published state-of-the-art heuristics or pure hyper-heuristics. Hence, the merit of the proposed representation compared with others remains to be evaluated. This paper is concerned with the more complex problem than that found in [4], as it considers a HFS-SDST-UPM problem. It introduces a mathematical formulation of the cardboard box shop (CBS), and introduces a lower bound on the optimum makespan. It extends the work in [4] by introducing and testing a number of variants of the meta-hyper-heuristic approach, pure hyper-heuristics, and two state-of-the-art meta-heuristics presented in [2] and [6], on instances generated using real world data.

The rest of the paper is organised as follows. The next section (Section 2) describes the CBS problem in detail and derives a lower bound on the optimum makespan. In Section 3, the proposed meta-hyper-heuristics are presented. Section 4 presents and discusses experimental results. Section 5 concludes the paper.

2 The problem

A production order specifies a number of boxes of a certain type to be produced. This is is referred to as a job. For practical reasons, concerning the difficulty to predict the time of arrival of input materials, the production is scheduled considering just those jobs for which the raw materials, white sheets and cardboard sheets, are available, i.e. it is assumed that all jobs to be scheduled are available at time 0. The products require four processing steps: (1) printing, (2) laminating, (3) cutting, and (4) folding-gluing. Each of these stages has its own storage of product in process, assumed to be infinite. Stage 4 (folding-gluing) is a special case consisting of 2 groups of machines. The first group has 4 machines that can process small and medium sized boxes, while the second group contains 2 identical machines that process large boxes. Since a box can be processed in one group only, folding-gluing was modelled as two independent stages: folding-gluing-1, with 4 machines, and folding-gluing-2, with 2 machines. Any job may be processed in just one of them. Printing, laminating and folding-gluing-2 have 2 identical machines each. Cutting and folding-gluing-1 have 7 and 4 unrelated parallel machines, respectively. The setup times are sequence dependent in printing and in folding-gluing-1 and folding-gluing-2. Preemptions are not allowed in any of the stages. This follows the fact that the setup times represent an important part of the total cost.

The cardboard box shop problem is denoted, using the $\alpha|\beta|\gamma$ notation from [7], as follows:

$$FH5(P2^{(1)}, P2^{(2)}, Q7^{(3)}, Q4^{(4)}, P2^{(5)}) \mid s_{sd}^{(1)}, s_{sd}^{(4)}, s_{sd}^{(5)} \mid C_{max},$$

where C_{max} is the makespan. Apart from the assumptions implied in such a description, we note the following: if any machine A is faster than any other machine B, and an operation a is to be assigned immediately after operation b, both the processing and the setup time of a will be faster if the assignment is to machine A instead of B regardless of the setup times being sequence dependent or not. This assumption is easy to justify since in stages where there are different types of the same machinery, new types are usually faster than the old ones with respect to both setup and processing.

In what follows, let $j = 1, \ldots, n$ and $k = 1, \ldots, m$ be the job and stage indices. The operation of job j to be processed in stage k is denoted by o_{jk}. Let $v_{kl} \in \mathbb{N}$ be the speed of machine l in stage k, i.e. the units of time required by machine l in stage k to process a unit of work or to complete a unit of setup work (both positive integers). Let p_{jk} represent the amount of work required by operation o_{jk}, therefore $p_{jkl} = \frac{p_{jk}}{v_{kl}}$ is the processing time it requires on machine l in stage k. Analogously, let set_up_{jqk} be the setup requirements for o_{jk} if it is processed immediately after o_{qk} and $set\_up_{jqkl} = \frac{set\_up_{jqk}}{v_{kl}}$ be the setup time required if o_{jk} is processed immediately after o_{qk} on machine l at stage k. Note that p_{jk} and set_up_{jqk} are given in units of required work, and p_{jkl} and set_up_{jqkl} are expressed in units of time. If $q = j$ set_up_{jqk} is the setup work required by operation o_{jk} if it is not preceded by any other operation.

Let $\hat{O}^{kl}$ be a set of operations o_{jk} assigned for processing to machine l in stage k. Let $S^{kl} = \{S^{kl}(1), \ldots, S^{kl}(|\hat{O}^{kl}|)\}$ be a sequence of the job indices of the operations in $\hat{O}^{kl}$, i.e. $S^{kl}(i)$ is the job index, j, of the operation o_{jk} assigned in the i^{th} place to machine l in stage k. Let S^k denote the set of sequences of job indices assigned to all the machines in stage k, $S^k = \bigcup_{l=1}^{M^{(k)}} S^{kl}$, where

$M^{(k)}$ is the number of machines in stage k. Let S denote the set of sequences assigned to all the machines, $S = \bigcup_{k=1}^{m} S^k$. The following conditions must hod for S to be feasible:

$$\bigcup_{l=1}^{M^{(k)}} \hat{O}^{kl} = O^k,\ k = 1, \ldots, m \tag{1}$$

where $O^k = \bigcup_j o_{jk}$, and

$$\bigcap_{l=1}^{M^{(k)}} \hat{O}^{kl} = \emptyset,\ k = 1, \ldots, m. \tag{2}$$

Constraints 1 and 2 guarantee that all operations are assigned for processing strictly once.

In addition, the feasibility condition includes precedence constraints imposed on the operations of jobs. S can be easily translated into a unique schedule by calculating the starting and completion times of operations. Let $c_{S^{kl}(i),k}$ be the completion time of the operation assigned in the i^{th} place to machine l in stage k calculated as follows:

$$c_{S^{kl}(i),k} = start_{S^{kl}(i),k} + p_{S^{kl}(i),k,l} + set\_up_{S^{kl}(i),S^{kl}(i-1),k,l},$$

where $S^{kl}(i-1)$ is the index of the operation preceding $o_{S^{kl}(i),k}$; $start_{S^{kl}(i),k}$ is the starting time of operation $o_{S^{kl}(i),k}$ calculated using the following recursive expression:

$$start_{S^{kl}(i),k} = \max\{c_{S^{kl}(i),k-1}, c_{S^{kl}(i-1),k}\}, \tag{3}$$

with the initial conditions: $start_{S^{kl}(1),1} = 0$, $start_{S^{kl}(1),k} = c_{S^{kl}(1),k-1}$ and $start_{S^{kl}(i),1} = c_{S^{kl}(i-1),1}$.

Expression 3 states that the starting time of an operation is calculated as the maximum between the completion time of its corresponding operation in the previous stage and the completion time of its precedent operation in the current stage.

Let ψ be a problem instance and Ω^ψ be the set of schedules S that satisfy (1) and (2). The problem can be formulated as:

$$\text{find} \qquad \min_{S \in \Omega^\psi} \max_j C_j(S),$$

where $C_j(S)$ is the completion time of job j, i.e. the time when job j exits the shop according to schedule S and $C_{max}(S) = \max_j C_j(S)$ is the makespan of schedule S. This means that, given a problem instance ψ, the task is to find a schedule $S \in \Omega^\psi$ that minimises the maximum completion time of operations, i.e. the makespan.

2.1 Lower bound on the optimum makespan for the HFS-SDST-UPM problem

In this section, a lower bound on the optimum makespan for the considered problem is presented. It will be used in forthcoming sections to evaluate the performance of the developed heuristics. This lower bound is an adaptation of the one presented in [8] for the HFS-SDST. It takes the maximum between two lower bounds: LB^1, which considers operations, and LB^2, which considers stages.

Let $\tilde{p}_{jkl} = p_{jkl} + \min_q set\_up_{jqkl}$ be the minimum possible total processing and setup time required by operation o_{jk} on machine l and let $\tilde{p}_{jk} = \min_l(p_{jkl} + \min_q set\_up_{jqkl})$ be the minimum possible processing and setup time required by operation o_{jk}. Then, an operation based lower bound is

$$LB^1 = \max_j \sum_{k=1}^{m} \tilde{p}_{jk},$$

which is the maximum of the minimum completion times of operations. This is clearly a lower bound since the makespan of a schedule cannot be smaller than the time required to process as fast as possible the job which requires the largest amount of processing and setup time.

Let $\tilde{r}_{jk} = \sum_{b=1}^{k-1} \tilde{p}_{jb}$ be the minimum possible release time of o_{jk}, i.e. the earliest possible starting time of operation o_{jk}. Let $\widetilde{tail}_{jk} = \sum_{b=k+1}^{m} \tilde{p}_{jb}$ be the minimum processing time that operation o_{jk} requires to be processed in stages $k+1$ to m. Let $r(k)$ and $tail(k)$ be the sequences of $\tilde{r}_{jk}$ and $\widetilde{tail}_{jk}$, $j = 1, \ldots, n$, values in the ascending order, so that $r(k)_1$ and $tail(k)_1$ are the $\min_j(\tilde{r}_{jk})$ and $\min_j(\widetilde{tail}_{jk})$ values, respectively. A stage based lower bound is as follows:

$$LB^2 = \max_k \left\{ r(k)_1 + \frac{\sum_{i=2}^{M^{(k)}} (r(k)_i - r(k)_1)}{M^{(k)}} + \frac{\sum_{j=1}^{n} (p_{jk} + \min_q set\_up_{jqk})}{\sum_{l=1}^{M^{(k)}} v_{kl}} + \frac{\sum_{i=1}^{M^{(k)}} tail(k)_i}{M^{(k)}} \right\}.$$

The first term in the braces is the minimum time that must pass before one of the machines in stage k starts processing. This happens when the first operation is released into stage k (after being processed in the previous stages). It may be the case that not all machines in stage k are activated at the same time. Since it is not clear when they become active, the required time by the second, third, up to the $M^{(k)}$th operation to arrive to stage k, is also taken into account. The second term considers the difference between the arrival time of these operations and the first one, and distributes this time onto the $M^{(k)}$ machines. Note that if $k = 1$ these two terms have a value of 0 (recall the assumption that all jobs are released at time 0 onto the first stage of the shop). The third term is the minimum time that stage k requires to process its workload. This is calculated as the minimum workload in stage k divided by the sum of the speeds of the machines in it. After completion in stage k, operations still have to be processed in the remaining stages. The fourth term adds the minimum possible time required by $M^{(k)}$ operations to be processed in stages posterior to k. This time is divided by the number of machines in stage k. Note that if $k = m$ this term has a value of 0. Summarising, LB^2 calculates, for every processing stage the following: (a) the minimum required time so that it starts processing, (b) the minimum time it requires to process its workload and (c) the minimum required time to finish the work in the remaining stages. The sum of (a), (b) and (c) is a stage based lower bound. Of interest is the maximum value among them.

The lower bound on the optimum makespan is calculated as

$$LB = \max\{LB^1, LB^2\}.$$

3 Solution approach

A GA based on a hybrid solution representation, motivated by the GA presented in [4], is developed for the stated problem. The first part of the chromosome, corresponding to the first stage of the shop, is a permutation $\pi = \pi(1), \ldots, \pi(n)$ of the job indices indicating the priority of the operations to be scheduled at the first stage of the shop floor. The second part, used for the rest of the shop, is a sequence $\mathcal{H} = \{h_1, \ldots, h_{|\mathcal{H}|}\}$ of dispatching rules. In total, 13 dispatching rules were adopted. Each of the dispatching rules works as follows: whenever a machine is idle, select the not yet scheduled and ready operation (released from previous stage) that best satisfies a certain criterion and assign it to the machine. Different criteria lead to different dispatching rules. The following ones were considered: *minimum release time*, *shortest processing time, longest processing time*, *less work remaining, more work remaining*, *earliest due date, latest due date*, *weighed shortest processing time, weighed longest processing time*, *lowest weighed work remaining, highest weighed work remaining*, *lowest weighed due date* and *highest weighed due date*.

A chromosome $(\pi, \mathcal{H})$ is evaluated by scheduling the stages in the following order: stage 1, stage 2, to stage m. Operations at stage 1 are prioritised and scheduled according to π. For the rest of the stages, the dispatching rules in $\mathcal{H}$ are called in order and used to schedule the. The fitness of the chromosome is the $C_{\max}$ value of the obtained schedule.

The hyper-heuristic representation (second part of the chromosome) is very flexible. One of its properties, which is exploited here to generate several variants of this method, is that it is possible to modify its size. See, for instance, the following representation:

$$\underbrace{\pi(1), \ldots, \pi(n)}_{\text{stage 1}}, \underbrace{h_1}_{\text{stage 2}}, \ldots, \underbrace{h_{m-1}}_{\text{stage } m}.$$

As in [4], the operations in the first stage of the shop are prioritised according to permutation π, in the second according to the dispatching rule h_1, in the third with h_2, and so on.

An also valid, but larger representation is:

$$\underbrace{\pi(1), \ldots, \pi(n)}_{\text{stage 1}}, \underbrace{h_1, h_2}_{\text{stage 2}}, \ldots, \underbrace{h_{t-1}, h_t}_{\text{stage } m},$$

where $t = 2 \times (m-1)$. The first stage of the shop is scheduled with π. In the second stage, the first $\frac{n}{2}$ operations are prioritised with h_1, the second half with h_2; in the third stage, the first half of operations are scheduled with h_3, the second half with h_4, and so on. Clearly, the second representation implies a larger search space than the first one, but it may also contain better solutions.

Similarly, chromosomes with 3, 4 and up to n dispatching rules per stage may be used. We investigate the performance of several variants of this approach, each with a different number of dispatching rules per stage.

3.1 GA operators and parameter tuning

Several GA operators and combinations of GA parameter values were considered. The selected ones, after the adequate testing, are briefly described next. Refer to [9] for further details.

For the permutation part of the chromosome, π, a 2–point order crossover (standard crossover operator for permutation representation) and a shift mutation were adopted. In the latter, an element of the chromosome is randomly selected and moved into a new randomly selected position, the rest of the genes are shifted as required to fill the empty space. For the heuristic representation, in order to do crossover, two chromosomes are selected as parents, and any gene h_i of the new chromosome is selected with a probability α from the fitter parent and 1-α from the other. In order to do mutation, $(n \times m) \times \beta$ randomly selected genes are substituted with a new randomly selected dispatching rule. Parameter β, $0 < \beta < 1$, controls the magnitude of the mutation. It has to be tuned.

All valid combinations of the following GA parameter values were considered. The combination shown in bold was the best performing after a pre-experimental phase with a stopping condition of 10,000 solution evaluations. **Population size** 50, 75, **100**; **selection mechanism**: **tournament selection 2** and 3 participants; **elitism** (keeping best individual found so far): **true**, false; **crossover probability**: 0.8, **0.9**, 0.95; **parameter** α: 0.3, **0.5**, 0.7; **mutation probability**: 0.05, **0.1**, 0.15; **parameter** β: 0.1, **0.05**, 0.025.

4 Computational experience

Several variants of the proposed GA, each in turn using a solution representation with 1 (as in [4]), 3, 6, 10, $\frac{n}{3}$, $\frac{n}{2}$, and n dispatching rules per stage, were evaluated. This section presents the results of the three best performing, having each in turn 1, 3 and 6 dispatching rules per stage. Refer to them as MHH_1, MHH_3 and MHH_6 (MHH standing for meta-hyper-heuristic).

Different than in previous work, [4], here, GAs with a pure hyper-heuristic representation are also evaluated. In this case, the combination of dispatching rules is used to schedule all the stages in the shop. As in the case of MHH, chromosomes with different number of dispatching rules per stage were considered. We present results with 10, 15 and n dispatching rules per stage. Refer to them as HH_{10}, HH_{15} and HH_n (HH standing for hyper-heuristic). These pure hyper-heuristics resemble the one presented in [10] for the solution of a real world hybrid job shop.

Two meta-heuristics tailored to solve similar problems to the HFS-SDST-UPM were also implemented. The first of these, presented in [6], is a GA with a Random Keys (RK) representation (real numbers representation), that was employed successfully for the HFS-SDST. The second is also a GA, presented in [2], denoted by GA_H (as originally proposed). This algorithm was designed to schedule a real world HFS-SDST with unrelated parallel machines. In each respective paper, the algorithms were tested against many other heuristics and compared favourably. Results of the eight heuristics in the set of instances described next will be reported.

4.1 Instance generation

The real world data was obtained from a company that produces cardboard boxes. We obtained and "cleaned" data collected throughout one year and generated from it a collection of around 1000 jobs with their corresponding $\tilde{p}_{jk}$ values.

Four types of problems with 30, 60, 90 and 120 jobs, 10 of each, (in total 40 instances) were generated. Each operation in every instance was randomly selected from the described collection of jobs. The setup times, due dates and weights of jobs were generated emulating those in the real shop floor.

4.2 Results

Each heuristic was run 5 times on each of the 40 instances with a stopping condition set to 10,000 solution evaluations on each run. The best solution found in the 5 runs was recorded. The deviation of the best solution found by heuristic a for problem ψ from the lower bound on the optimum makespan (presented in Section 2.1) was calculated as:

$$percentage\_error(a, \psi) = 100 \times \frac{C_{max}(a, \psi) - LB(\psi)}{LB(\psi)}.$$

The mean of the *percentage_error* values achieved by each heuristic on each set of instances, grouped according to their size, are presented in Table 1. The best values are shown in bold. We may notice that in each of the 4 groups of instances, and overall, the MHH variants perform better than the rest, suggesting that the combination of meta-heuristic with hyper-heuristic, as in MHH, is superior to any of them separately.

Non-parametric statistics confirm the superiority of MHH. Figure 1 shows the 0.95 confidence interval of the mean of the ranks of the studied heuristics according to their performance, i.e. achieved makespan. It can be observed that the rank of the algorithms is as follows: in the first place are the MHH methods, followed by RKGA and GA_H and last are the "pure" hyper-heuristics.

Table 1: Mean percentage error from LB (best in bold)

	number of jobs (n)				
	30	60	90	120	overall
HHS_{10}	9.77	17.94	14.95	10.88	13.38
HHS_{15}	9.05	17.22	15.06	11.05	13.09
HHS_n	8.85	17.15	15.98	9.88	12.96
RKGA	7.25	14.12	12.37	7.78	10.38
GA_H	7.73	13.32	10.99	6.99	9.76
$MHHS_1$	**6.39**	10.36	8.18	5.16	**7.52**
$MHHS_3$	6.48	**10.11**	8.60	5.25	7.61
$MHHS_6$	6.83	**10.11**	**8.29**	**5.13**	7.59

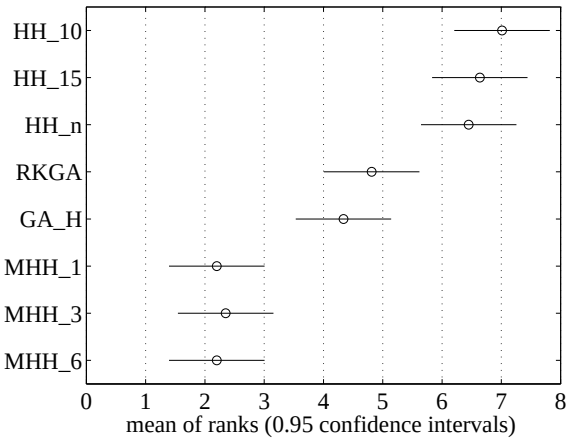


Figure 1: Mean of the ranks of the heuristics according to their performance minimising makespan

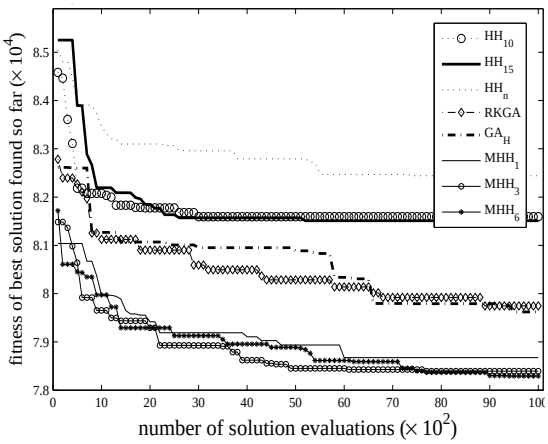


Figure 2: Typical convergence process of the studied heuristics

The superiority of MHH is probably due to the fact that it exploits knowledge that is implicit in the design of RKGA and GA_H. These two algorithms also use permutation-like representations to schedule the first stage of the shop and a constructive procedure to generate the rest. Such a definition of the search space in which all possible permutations of jobs are allowed in the first stage, while the remaining stages allow subsets of all possible permutations determined by constructive heuristics (i.e. dispatching rules) is proved to be very effective for the HFS problem, [11]. In MHH, a combination of dispatching rules that best suits the instance in hand substitutes the constructive procedure. This has the effect of increasing the search space and potentially allowing access to better quality solutions.

The convergence process of the studied heuristics was also investigated. The plot showing the *convergence curves*, typical of a large number of instances, is shown in Figure 2. The curves are similar for all heuristics. However, the MHH methods start their search at a more favourable point. Since all heuristics, except GA_H, keep a population of the same size, this difference in the performance is attributed to the MHH representation. The good performance of MHH can, therefore, be attributed to its representation allowing the GA to search more promising areas than the other heuristics.

5 Conclusion

The presented research was motivated by a real world Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Parallel Machines, (HFS-SDST-UPM). Several variants of a Genetic Algorithm (GA) with a hybrid meta-heuristic with hyper-heuristic representation have been developed. Each of the proposed variants combines a permutation representation with a sequence of dispatching rules. Different numbers of dispatching rules in the sequence lead to different variants of the approach.

The proposed methods were compared with pure hyper-heuristics and two state-of-the-art meta-heuristics. Computational results reveal that the proposed approaches are superior to their competitors. It is safe to conclude, therefore, that the hybrid GA representation exploits the strengths of both meta-heuristics and hyper-heuristics in a synergistic manner.

References

[1] J.N.D. Gupta (1988), Two-stage hybrid flow shop scheduling problem, *Operational Research Society* **39**, 359 – 364.

[2] R.R. García and C. Maroto (2006), A genetic algorithm for hybrid flow shops with sequence dependent setup times and machine elegibility, *European Journal of Operational Research* **169**, 781 – 800.

[3] A. Allahverdi and F.S. Al-Anzi (2006), Scheduling multi-stage parallel-processor services to minimize average response time, *Journal of the Operational Research Society* **57**, 101 – 110.

[4] J.A.V. Rodríguez and A. Salhi (2007), A robust meta-hyper-heuristic approach to hybrid flow shop scheduling, in K.P. Dahal, K.C. Tan and P.I. Cowling, editors, *Evolutionary Scheduling*, Springer.

[5] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg (2003), Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, 457 – 474, Springer.

[6] M.E. Kurz and R. G. Askin (2003), Scheduling flexible flow lines with sequence dependent set-up times, *European Journal of Operational Research* **159**, 66 – 82.

[7] A. Vignier, J.C. Billaut, and C. Proust (1999), Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art, *RAIRO Recherche opérationnelle* **33**, 117 – 183.

[8] M.E. Kurz and R.G. Askin (2001), An adaptable problem-space-based search method for flexible flow line scheduling, *IIE Transactions* **33**, 691 – 693.

[9] J.A.V. Rodríguez (2007), Meta-hyper-heuristics for hybrid flow shops, Ph.D. thesis, University of Essex.

[10] C. Fayad and S. Petrovic (2005), Genetic algorithm for the real world fuzzy job-shop scheduling, In *Proceedings of IEA/AIE-2005, Lecture Notes in Computer Science, Vol. 3533*, 524 – 533, Springer-Verlag.

[11] D.L. Santos, J.L. Hunssucker, and D.E. Deal (1995), FLOWMULT: Permutation sequences for flow shops with multiple processors, *Journal of Information and Optimization Sciences* **16**, 351 – 366.

Competitive Agent Scheduling with Controllable Processing Times

Guohua Wan
College of Management, Shenzhen University, Shenzhen 518060, China, gh_wan@china.com
Joseph Y.-T. Leung
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, U.S.A., leung@oak.njit.edu
Michael Pinedo
Stern School of Business, New York University, 44 West Fourth Street, New York, NY 10012, U.S.A., mpinedo@stern.nyu.edu

We consider several competitive agent scheduling problems with controllable processing times, where two agents A and B compete for a single machine to process their jobs. The objective function for agent B is always the same, namely $f_{\max}$. Several different objective functions are considered for agent A, including the total compression cost subject to deadline constraints (the imprecise computation model), the total flow time plus compression cost, the maximum tardiness plus compression cost and the maximum lateness plus compression cost. These problems have various applications in computer systems as well as in operations management. We provide NP-hardness proofs for the more general problems and polynomial time algorithms for several special cases of the problems.

Keywords: Agent Based Scheduling, Single Machine, Controllable processing times, Availability constraints, Imprecise computation, Total flow time; Maximum tardiness, Maximum lateness.

1 Introduction

We consider several competitive agent scheduling problems where two sets of jobs N_1 and N_2 (belonging to agents A and B, respectively) have to be processed on a single machine. Agent A has to schedule n_1 jobs in N_1 and agent B has to schedule n_2 jobs in N_2. Let n denote the total number of jobs, i.e., $n = n_1 + n_2$. The processing time, release date and due date of job $j \in N_1$ (N_2) are denoted by p_j^a, r_j^a and d_j^a (p_j^b, r_j^b and d_j^b), respectively. Unlike the classical model in deterministic scheduling where all processing times are fixed and known in advance, the processing times are in what follows controllable and can be chosen by the decision maker. In this paper, we assume that agent A's jobs are controllable, while agent B's jobs are not. Formally, for each job $j \in N_1$, there is a maximum value of the processing time $\overline{p}_j^a$ which can be compressed to a minimum value $\underline{p}_j^a$ ($\underline{p}_j^a \leq \overline{p}_j^a$). Compressing $\overline{p}_j^a$ to some actual processing time $p_j^a \in [\underline{p}_j^a, \overline{p}_j^a]$ may decrease the job completion time, but incurs an additional cost $w_j^a x_j^a$, where $x_j^a = \overline{p}_j^a - p_j^a$ is the amount of compression of job $j \in N_1$ and w_j^a is the compression cost per unit time. The total compression cost is represented by a linear function $\sum_{j \in N_1} w_j^a x_j^a$. We consider the optimization problem in which the value of the objective function of agent A has to be minimized, while the value of the objective function of agent B must be kept at less than or equal to a fixed value Q.

The classical notation for machine scheduling is based on a triplet $\alpha \mid \beta \mid \gamma$. Agnetis et al. (2004) extend this notation for the two agent problem to $\alpha \mid \beta \mid \gamma^a : \gamma^b$. Their optimization problems can be described as follows: Given that agent B keeps the value of its objective function γ^b less than or equal to Q, agent A has to minimize the value of its objective function γ^a. In this paper, we may assume that either one set or both sets of jobs have different release dates and that

either one set or both sets of jobs are subject to preemptions. If the jobs of both agents are subject to the same processing restrictions and constraints (as in Agnetis et al. (2004)), then the single machine problem will be referred to as $1 \mid \beta \mid \gamma^a : \gamma^b$. If the processing restrictions and constraints of agent A's jobs are different from the processing restrictions and constraints of agent B's jobs, we refer to the single machine problem as $1 \mid \beta^a : \beta^b \mid \gamma^a : \gamma^b$.

Scheduling models with competitive agents have already received some attention in the literature. Baker and Smith (2003) and Agnetis et al. (2004) consider single machine scheduling problems with two agents in which all jobs of the two sets are released at time 0 and both sets of jobs are subject to the same processing restrictions and constraints. The objective functions considered in their research include the total weighted completion time ($\sum w_j C_j$), the number of tardy jobs ($\sum U_j$) and the maximum of regular functions ($f_{\max}$). Leung et al. (2006) consider a scheduling environment with $m \geq 1$ identical machines in parallel and two agents, and generalize the results of Baker and Smith (2003) and Agnetis et al. (2004) by including the total tardiness objective, allowing for preemptions, and considering jobs with different release dates.

Scheduling models with controllable processing times have received considerable attention in the literature, see, for example, the survey by Nowicki and Zdrzalka (1990). The study of these models is motivated by their various applications to production and operations management, computer systems, among others. The main issue here is to establish a trade-off between job completion times and the costs of compression.

Studies of scheduling problems with controllable processing times were initiated by Vickson (1980a, 1980b). The problem introduced by Vickson has attracted a lot of attention, see, for example, Hoogeveen and Woeginger (2002), Janiak et al. (2005), Wan et al. (2001), Janiak and Kovalyov (1996), Nowicki and Zdrzalka (1990, 1995), Shakhlevich and Strusevich (2005), and van Wassenhove and Baker (1982), among others. In the triplet notation, we put *ctrl* in the second field to refer to controllable processing times.

In the following, we first state the problems, then we give NP-hardness proofs of the problems, and describe algorithms for solving the problems or their spacial cases. We conclude and discuss some future research directions in the last section.

2 Problem Description

In the scheduling problems considered in this paper, each job j of agent B has a penalty function $f_j^b(C_j^b)$ and the objective function of agent B is simply $f_{\max}^b = \max(f_1^b(C_1^b), \ldots, f_{n_2}^b(C_{n_2}^b))$. Given that agent B keeps the value of $f_{\max}^b$ less than or equal to Q, agent A has to minimize the value of one of the following objective functions.

(1) In the first problem each job j of agent A must meet a deadline $\bar{d}_j^a$. Our goal is to determine the actual processing times of agent A's jobs so that the total compression cost is minimized. It is referred to as imprecise computation model with controllable processing times (see Leung (2004)). Using the notation introduced above, we denote the problems by $1 \mid ctrl^a, r_j^a, \bar{d}_j^a, pmtn^a : pmtn^b \mid \sum w_j^a x_j^a : f_{max}^b$ and $1 \mid ctrl^a, r_j^a, \bar{d}_j^a, pmtn^a : \circ^b \mid \sum w_j^a x_j^a : f_{max}^b$. We also consider the case where all the jobs of both agents A and B are released at time 0; i.e., $1 \mid ctrl^a, \bar{d}_j^a, pmtn^a : \circ^b \mid \sum w_j^a x_j^a : f_{max}^b$.

(2) The second problem considered is to minimize the total flow time plus job compression costs. Again, the jobs of agent A may be preempted. We shall show that when the jobs of agent A have different release dates, the problem is unary NP-hard; i.e., the problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid \sum (C_j^a + w_j^a x_j^a) : f_{max}^b$ is unary NP-hard. However, if the jobs of agent A all have the same release date, then the complexities of the nonpreemptive and preemptive cases are the same; i.e., the problem $1 \mid ctrl^a : \circ^b \mid \sum (C_j^a + w_j^a x_j^a) : f_{max}^b$ and the problem $1 \mid ctrl^a, pmtn^a : \circ^b \mid \sum (C_j^a + w_j^a x_j^a) :$

f^b_{max} have the same complexity. Although we do not know the complexity of these two problems, we are able to give a polynomial-time algorithm for the special case when the jobs have agreeable weights, i.e., when $p^a_1 \le p^a_2 \le \cdots \le p^a_{n_1}$ and $w^a_1 \le w^a_2 \le \cdots \le w^a_{n_1}$.

(3) The third and the fourth problem concern the minimization of the maximum tardiness plus job compression cost and the minimization of the maximum lateness plus job compression cost, respectively. The jobs of agent A may again be preempted. If the jobs of agent A have arbitrary release dates, then both problems are unary NP-hard; i.e., the problems $1 \mid ctrl^a, r^a_j, pmtn^a : \circ^b \mid (T_{max} + \sum w^a_j x^a_j) : f^b_{max}$ and $1 \mid ctrl^a, r^a_j, pmtn^a : \circ^b \mid (L_{max} + \sum w^a_j x^a_j) : f^b_{max}$ are both unary NP-hard. When the jobs of agent A have the same release date, then the complexities of the nonpreemptive and preemptive cases are identical. Although we do not know the complexity of these two problems, we are again able to provide a polynomial-time algorithm for the special case when the jobs have agreeable weights, i.e., when $d^a_1 \le d^a_2 \le \cdots \le d^a_{n_1}$ and $w^a_1 \le w^a_2 \le \cdots \le w^a_{n_1}$.

The problems described above may find various applications in computer systems as well as in operations management. For instance, in computer networks a server may service several classes of jobs such as file downloading, voice messaging and web browsing, where one class of jobs may have a high priority and another class may have a lower priority. A request to the server for voice messaging or web browsing, constitutes a job. Jobs may have various characteristics such as release dates, due dates, and/or preemption. The server may put the jobs into two classes, say, one for web browsing and the other for voice messaging. In order to provide a satisfactory quality of service, it is necessary to keep on the one hand the maximum penalty of jobs for web browsing less than or equal to some fixed value, and, on the other hand, meet the deadlines of the voice messaging packages. To keep the voice quality at a satisfactory level, it is desirable to discard as few packages as possible, i.e., to minimize the total amount of compression of jobs for voice messaging. This application can be modeled by (1).

3 Imprecise Computation

We first consider the problem $1 \mid ctrl^a, r^a_j, \bar{d}^a_j, pmtn^a : \circ^b \mid \sum w^a_j x^a_j : f^b_{max}$. We show that it is unary NP-hard via a reduction from 3-PARTITION, which is known to be unary NP-hard (see Garey and Johnson (1979)).

3-PARTITION: Given positive integers $a_1, \ldots, a_{3n}$ and b with $\frac{b}{4} < a_j < \frac{b}{2}, j = 1, \ldots, 3n$ and $\sum_{j=1}^{3n} a_j = nb$, do there exist n pairwise disjoint three element subsets $S_i \subset \{1, \ldots, n\}$ such that $\sum_{j \in S_i} a_j = b, i = 1, \ldots, n$?

Theorem 1. The problem $1 \mid ctrl^a, r^a_j, \bar{d}^a_j, pmtn^a : \circ^b \mid \sum w^a_j x^a_j : f^b_{max}$ is unary NP-hard.
Proof. The proof is done via a reduction from 3-PARTITION. □

However, if preemptions are allowed for the jobs of agent B, then the problem is solvable in polynomial time. That is, $1 \mid ctrl^a, r^a_j, \bar{d}^a_j, pmtn^a : pmtn^b \mid \sum w^a_j x^a_j : f^b_{max}$ is solvable in polynomial time. The algorithm is based on the polynomial-time algorithm for minimizing the total weighted error in the imprecise computation model that is due to Leung et al. (1994). The algorithm of Leung-Yu-Wei solves the problem $1 \mid r_j, \bar{d}_j, pmtn \mid \sum w_j x_j$ in $O(n \log n + kn)$ time, where n is the number of jobs and k is the number of distinct values of $\{w_j\}$.

Algorithm 1.
Step 1: For each job j of agent B, compute a "deadline" $\bar{d}^b_j$ via $f_{\max} \le Q$ (assuming f^{-1} can be computed in constant time). Let the release date of job j of agent B be $r^b_j = 0$ and the weight be

$w_j^b = 1 + \max_{1 \le j \le n_1}\{w_j^a\}$, for $j = 1, ..., n_2$. Furthermore, let the jobs of agent B be uncompressible (i.e., $\overline{p}_j^b = \underline{p}_j^b$).

Step 2: Use the algorithm of Leung-Yu-Wei to generate a schedule of all jobs of agents A and B together.

Remark 1. The time complexity of Algorithm 1 is $O((n_1+n_2)\log(n_1+n_2)+(k+1)(n_1+n_2))$, where k is the number of distinct weights of jobs of agent A.

Remark 2. Algorithm 1 can be generalized to solve the problem $1 \mid ctrl^a, r_j^a, \bar{d}_j^a, pmtn^a : r_j^b, pmtn^b \mid \sum w_j^a x_j^a : f_{max}^b$ as well. In Step 1 of Algorithm 1, we simply let the release date of job j of agent B be r_j^b.

Remark 3. Algorithm 1 can be generalized to solve the problem $1 \mid ctrl^a, \bar{d}_j^a, pmtn^a : \circ^b \mid \sum w_j^a x_j^a : f_{max}^b$ as well. This is because of Property 1 shown below.

Property 1. If $r_j^a = 0,\ \forall j$, then there exists an optimal schedule in which jobs of agent B are scheduled as late as possible, i.e., against their "deadlines" (computed via $f_{\max}^b \le Q$).

Proof. First, note that in an optimal schedule, all the jobs of agent B must satisfy $f_{\max}^b \le Q$. Suppose that for a job k of agent B, $f_k^b < Q$, then we can always move it backwards so that $f_k^b = Q$ (or reaching another job of agent B) and at the same time move some pieces of jobs of agent A forward. Clearly, this will not violate the constraint $f_{\max}^b \le Q$ and it will not increase $\sum w_j^a x_j^a$. □

Using Property 1, we can develop our algorithm as follows. We schedule all the jobs of both agents together using the Leung-Yu-Wei algorithm. Although in the schedule created jobs of agent B may be preempted, we can always combine the pieces of a job of agent B together and move it towards its "deadline" $\bar{d}_j^b$, by Property 1. In this way, we can always convert a schedule where jobs of agent B may be preempted into one where jobs of agent B are not preempted.

Remark 4. Algorithm 1 can be generalized to solve the problem $1 \mid ctrl^a, \bar{d}_j^a : \circ^b \mid \sum w_j^a x_j^a : f_{max}^b$ as well. We first solve the problem $1 \mid ctrl^a, \bar{d}_j^a, pmtn^a : \circ^b \mid \sum w_j^a x_j^a : f_{max}^b$. We then merge the preempted pieces of jobs of agent A by moving the jobs of agent B forward. Clearly, this will not violate the constraints of the jobs of either agent.

4 Total flow time plus compression cost

We now turn to the problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid \sum(C_j^a + w_j^a x_j^a) : f_{max}^b$. We will show that it is unary NP-hard via a reduction from 3-PARTITION.

Theorem 2. The problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid \sum(C_j^a + w_j^a x_j^a) : f_{max}^b$ is unary NP-hard.

Proof. The proof is done via a reduction from 3-PARTITION. □

We now consider the problem when the jobs of agent A are all released at time 0. At present, the complexity of this problem is not known. However, we do know that the complexities of the nonpreemptive case and the preemptive case are the same; i.e., the problem $1 \mid ctrl^a, pmtn^a : \circ^b \mid \sum(C_j^a + w_j^a x_j^a) : f_{max}^b$ and the problem $1 \mid ctrl^a : \circ^b \mid \sum(C_j^a + w_j^a x_j^a) : f_{max}^b$ have the same complexity. This is due to Property 1 described in Section 3.

Although the complexity of the general case is not known, we are able to provide a polynomial-time algorithm for the special case with agreeable weights, i.e., when $p_1^a \le p_2^a \le \cdots \le p_{n_1}^a$ and $w_1^a \le w_2^a \le \cdots \le w_{n_1}^a$.

We use the following terminology in the algorithm. Job j of agent A is referred to as "compressed" if $p_j^a = \underline{p}_j^a$, i.e., it is fully compressed and cannot undergo any additional compression. Job j of agent A is referred to as "uncompressed" if $p_j^a > \underline{p}_j^a$; it can undergo some (additional) compression. Job 0 with $p_0^a = 0$ is a dummy job that belongs to Agent A and that is scheduled all the way at the beginning. Job 0 is considered "compressed".

Algorithm 2. (A polynomial-time algorithm for a special case)

Step 1: For each job j of agent B, compute its "deadline" $\bar{d}_j^b$ via $f_{\max}^b \leq Q$. Starting from $\max_{1\leq j\leq n_2}\{\bar{d}_j^b\}$, schedule the jobs of agent B backwards so that each job is scheduled as close to its deadline as possible.

Step 2: Define block i as the i^{th} set of contiguously processed jobs of agent B. Let there be $k_1 \leq n_2$ blocks, and let h_i be the length of block i, $1 \leq i \leq k_1$.

Step 3: For each job j of agent A, let $p_j^a = \overline{p}_j^a$. Let $p_0^a = 0$.

Step 4: Let $P = \{p_j^a \mid 1 \leq j \leq n_1\}$. ($P$ is the current set of processing times of agent A's jobs.) Set *Cost*= 0 and *Benefit*= 0.

Step 5: Schedule the jobs of agent A by the preemptive SPT rule, skipping the time slots occupied by jobs of agent B. Let l_i, $1 \leq i \leq k_2$, be the length of the job piece of agent A immediately following block i. Note that $k_2 \leq k_1$.

Step 6: Let z be the first "uncompressed" job; i.e., the first $z-1$ jobs are fully compressed. If every job is "compressed" then go to Step 9. Let q be the first block of agent B's jobs that appears after the start time of job z. Let $l_{\min} = \min_{q\leq i\leq k_2}\{l_i\}$. Let $x_z = p_z^a - \underline{p}_z^a$ and $y_z = p_z^a - p_{z'}^a$, where z' is the largest index such that $p_{z'}^a < p_z^a$. Let $\delta = \min\{x_z, y_z, l_{\min}\}$.

Step 7: Compress job z by an amount δ; i.e., $p_z^a = p_z^a - \delta$. We consider the following three cases, in the order they are presented.

[Case 1.] ($\delta = l_{\min}$) *Cost* = *Cost* $+w_z * \delta$, *Benefit* = *Benefit* $+(n_1 - z + 1) * \delta + \sum_{q\leq j\leq k_2} u_j h_j$, where u_j is the number of jobs that jump over block j (meaning that they complete after block j, before compressing job z and complete before block j afterwards).

[Case 2.] ($\delta = x_z$) *Cost*= *Cost* $+w_z * \delta$, *Benefit*= *Benefit* $+(n_1 - z + 1) * \delta$, mark job z as "compressed".

[Case 3.] ($\delta = y_z$) *Cost* = *Cost* $+w_z * \delta$, *Benefit* = *Benefit* $+(n_1 - z + 1) * \delta$. Reindex the jobs of agent A in ascending order of their processing times, where job z will appear before any job with the same processing time as job z.

Step 8: If *Cost* < *Benefit*, then go to Step 4 else go to Step 5.

Step 9: Using P (which has the last recorded processing times of the jobs), we schedule the jobs of agent A by the preemptive SPT rule, skipping the time slots occupied by jobs of agent B. This is the final schedule. □

Theorem 3. Algorithm 2 solves the problem $1 \mid ctrl^a, pmtn^a : \circ^b \mid \sum(C_j^a + w_j^a x_j^a) : f_{max}^b$ with agreeable weights in $O(n_2(n_1+n_2)n_1^2 \log n_1)$ time.

Proof. Since the weights are agreeable, the algorithm always compresses the first compressible job before it proceeds to compress the next compressible job. Thus it maintains the agreeability of the remaining compressible jobs, until all the compressible jobs are compressed (provided the compressions are beneficial). This guarantees the correctness of the algorithm.

Furthermore, note that in the above algorithm, Steps 1 and 2 take at most $O(n_2 \log n_2)$ time. Steps 3 and 4 take $O(n_1)$ time. Steps 5 and 6 take $O(n_1 \log n_1)$ time. Step 7 takes at most $O(n_1)$ time and Step 8 takes constant time. The most expensive time of the algorithm is the loop consisting of Step 5 to Step 8. In the worst case, the algorithm will terminate when every job is "compressed". In Step 7, Case 2 will compress a job, while Cases 1 and 3 will not. Thus, if we can

determine the number of steps Cases 1 and 3 take before Case 2 is executed, then we can determine the running time of the algorithm. Case 3 takes at most $r = O(n_1)$ steps. Case 1 takes at most $O(n_2(n_1+n_2))$ steps. This is because there could be at most n_1+n_2 pieces of jobs of agent A (The jobs of agent A are preempted at most n_2 times). In the worst case, these pieces all appear after the last block of agent B's jobs. For each execution of Case 1, one piece will be moved to the left. Therefore, after $O(n_2(n_1+n_2))$ steps, every piece of agent A's jobs will be moved before the first block of agent B's jobs, and then Case 1 cannot occur again. This means that we have to execute Case 2 (which will compress the job). So the loop will be executed at most $O(n_2(n_1+n_2))$ steps for each job. Thus, the overall running time of Algorithm 2 is $O(n_2(n_1+n_2)n_1^2 \log n_1)$. □

Remark 5. For the problem $1 \mid ctrl^a : \circ^b \mid \sum(C_j^a + w_j x_j^a) : f_{max}^b$ with agreeable weights, we can still run the above algorithm to obtain a schedule with preemption first. Then we merge the preempted pieces of jobs of agent A by moving the jobs of agent B forward. Obviously this will neither violate the constraint for jobs of agent B nor increase the total cost of agent A.

5 Maximum tardiness plus compression cost

In this section, we consider two problems involving due dates, namely, the objective function of agent A is the maximum tardiness (or maximum lateness) plus total compression cost. We shall prove that if the jobs of agent A have different release dates both these problems are unary NP-hard. First, we consider the problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid (T_{max} + \sum w_j^a x_j^a) : f_{max}^b$.

Theorem 4. The problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid (T_{max} + \sum w_j^a x_j^a) : f_{max}^b$ is unary NP-hard.

Proof. The proof is done via a reduction from 3-PARTITION. □

Remark 6. Using the same proof as above, we can show that the problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid (L_{max} + \sum w_j^a x_j^a) : f_{max}^b$ is also unary NP-hard.

At present, the complexity of the problem is not known if the jobs of agent A are all released at time 0. However, if $r_j^a = 0$ and $d_i^a \le d_j^a \Rightarrow w_i^a \le w_j^a$ for all i and j, then the problem becomes polynomially solvable. The algorithm is based on the fact that the jobs of agent A will appear in the optimal schedule in EDD order for any set of processing times of these jobs.

Property 2. There exists an optimal sequence for the problem in which all the jobs of agent A are sequenced by the modified preemptive EDD rule.

Because of this property, we can always compress the processing times of these jobs without changing the optimal job sequence of agent A. Below we describe such a polynomial-time algorithm for this problem. In the algorithm, we schedule jobs of agent B first and against their "deadlines" (the "deadlines" are computed via $f_{max}^b \le Q$ for all the jobs of agent B).

Algorithm 3. (A polynomial-time algorithm for a special case)

Step 0: Sort the jobs of agent A in increasing order of their due dates. Assume after sorting, we have $d_1^a \le d_2^a \le \cdots \le d_{n_1}^a$.

Step 1: For each job j of agent B, compute its "deadline" $\bar{d}_j^b$ via $f_{\max}^b \le Q$. Starting from $\max_{1 \le j \le n_2}\{\bar{d}_j^b\}$, schedule the jobs of agent B backwards so that each job is scheduled as close to its deadline as possible.

Step 2: Define block i as the i^{th} set of contiguously processed jobs of agent B. Let there be $k_1 \le n_2$ blocks, and let h_i be the length of block i, $1 \le i \le k_1$.

Step 3: For each job j of agent A, let $p_j^a = \overline{p}_j^a$. For each job j of agent A, mark job j as "uncompressed" if $\overline{p}_j^a > \underline{p}_j^a$; otherwise, mark job j as "compressed".

Step 4: Let $P = \{p_j^a \mid 1 \leq j \leq n_1\}$ (P is the current set of processing times of agent A's jobs). $Cost = 0$ and $Benefit = 0$.

Step 5: Schedule the jobs of agent A by the preemptive EDD rule, skipping the time slots occupied by jobs of agent B. Let the maximum tardiness be T_{max} and $j_{max} = \min\{j : T_j = T_{max}\}$. If $T_{max} = 0$, go to Step 10.

Step 6: Let block k be last block before starting of job j_{max} and let the length of the block be h_k. Let l be the distance from time $C_{j_{max}}$ to the end of block k. If there is no such a block, let $l = 0$ and $h = 0$. Furthermore, let $T_{next} = \max\{T_j : \text{job } j \text{ is before job } j_{max}\}$ and $j_{next} = \min\{j : T_j = T_{next}\}$. If there is no such job, then $T_{next} = 0$.

Step 7: Let z be the first "uncompressed" job; i.e., the first $z-1$ jobs are fully compressed. If every job before job j_{max} and including job j_{max} is "compressed" then go to Step 10. Let $x_z = p_z^a - \underline{p}_z^a$ and $y = T_{max} - T_{next}$.

Step 8: Let $\delta = \min\{x_z, y, l\}$. Compress job z by δ amount; i.e., $p_z^a = p_z^a - \delta$. We consider the following three cases, in the order they are presented.

[Case 1.] ($\delta = l$) $Cost = Cost + w_z * \delta$, $Benefit = Benefit + T_{max} - \max\{T_j'\}$, where T_j' is the new tardiness of jobs between job z and block k after compressing job z by l amount.

[Case 2.] ($\delta = x_z$) $Cost = Cost + w_z * \delta$, $Benefit = Benefit + \delta$, mark job z as "compressed".

[Case 3.] ($\delta = z$) $Cost = Cost + w_z * \delta$, $Benefit = Benefit + T_{max} - \max\{T_j'\}$, where T_j' is the new tardiness of jobs between job z and block k after compressing job z by y amount.

Step 9: If $Cost < Benefit$ then go to Step 4 else go to Step 5.

Step 10: Using P (which has the last recorded processing times of the jobs), we schedule the jobs of agent A according to the preemptive EDD rule, skipping the time slots occupied by jobs of agent B. This is the final schedule.

Theorem 5. Algorithm 3 solves the problem $1 \mid ctrl^a, r_j^a, pmtn^a : \circ^b \mid (T_{max} + \sum w_j^a x_j^a) : f_{max}^b$ with agreeable weights in $O(n_2(n_1+n_2)n_1^2)$ time.

Proof. The correctness part of the proof is similar to that in the proof of Theorem 3.

The computational complexity is similar as well, except it does not need to sort the jobs of agent A in each iteration due to Property 2. □

Remark 7. If $d_i^a \leq d_j^a \Rightarrow w_i^a \leq w_j^a$ for all i and j, then a polynomial-time algorithm for the problem $1 \mid ctrl^a, pmtn^a : \circ^b \mid (L_{max}^a + \sum w_j x_j^a) : f_{max}^b$ is similar to that for the problem $1 \mid ctrl^a, pmtn^a : \circ^b \mid (T_{max}^a + \sum w_j x_j^a) : f_{max}^b$ with agreeable weights, except that in Step 5, it is not necessary to test if $L_{max} = 0$. The time complexity is the same as well.

Remark 8. We can generalize Algorithm 3 to solve the problem $1 \mid ctrl^a : \circ^b \mid (T_{max}^a + \sum w_j x_j^a) : f_{max}^b$ and the problem $1 \mid ctrl^a : \circ^b \mid (L_{max}^a + \sum w_j x_j^a) : f_{max}^b$ as well. We first solve the preempted version of the problem and then merge the preempted pieces of the jobs of agent A to obtain a nonpreemptive schedule.

6 Concluding remarks

We have studied several competitive agent scheduling problems with controllable processing times, where processing times of jobs of agent A are controllable. We provided either NP-hardness proofs or polynomial-time algorithms for some special cases of the problems. In fact, these results can

be extended to the corresponding cases where the processing times of jobs of agent B are also controllable with the compression costs charged to Agent A.

All the results obtained in this paper for two agent models in which the jobs of Agent A are allowed to be preempted have corresponding results in preemptive single machine scheduling environments with a single agent and availability constraints. The time periods that the machine in this paper was reserved for Agent B were always "against the deadlines of the jobs of Agent B". The periods set aside for the processing of the jobs of Agent B were basically periods that the machine was not available for the processing of jobs of Agent A. There is an extensive amount of research done on scheduling with availability constraints, see Lee (2004). However, to our knowledge, single machine scheduling with availability constraints and controllable processing times had not yet been considered in the literature.

For future research, we note that the computational complexity of the problems with total flow time and maximum tardiness/lateness of agent A remain open. For those problems that are unary NP-hard, it will be interesting to develop approximation algorithms. Furthermore, it will be of interest to consider problems in a parallel machine environment.

Acknowledgment: We thank two anonymous referees for their helpful comments. The work of the first author is supported in part by the NSF of China Grant (70372058) and Guangdong (China) NSF Grant (031808). The work of the second author is supported in part by the NSF under Grants DMI-0300156 and DMI-0556010. The work of the third author is supported in part by the NSF under Grants DMI-0555999.

References

[1] A. Agnetis , P. B. Mirchandani, D. Pacciarelli and A. Pacifici (2004), Scheduling Problems with Two Competing Agents, *Operations Research* **52**, 229 – 242.

[2] K. R. Baker and J. C. Smith (2003), A Multiple-Criterion Model for Machine Scheduling, *Journal of Scheduling* **6**, 7 – 16.

[3] M. R. Garey and D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA.

[4] H. Hoogeveen and G. J. Woeginger (2002), Some Comments on Sequencing with Controllable Processing Times, *Computing* **68**, 181 – 192.

[5] A. Janiak and M. Y. Kovalyov (1996), Single Machine Scheduling Subject to Deadlines and Resource Dependent Processing Times, *Eur. J. Oper. Res.* **94**, 284 – 291.

[6] C-Y. Lee (2004), Machine Scheduling with Availability Constraints. In J. Y-T. Leung (ed.) *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, CRC Press, Boca Raton, FL.

[7] J. Y-T. Leung (2004), Minimizing Total Weighted Error for Imprecise Computation Tasks and Related Problems, In J. Y-T. Leung (ed.) *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, CRC Press, Boca Raton, FL.

[8] J. Y-T. Leung, M. L. Pinedo and G. Wan (2006), Competitive Agent Scheduling and Its Applications. Submitted.

[9] J. Y-T. Leung, V. K. M. Yu and W.-D. Wei (1994), Minimizing the Weighted Number of Tardy Task Units, *Discrete Appl. Math.* **51**, 307 – 316.

[10] E. Nowicki and S. Zdrzalka (1990), A Survey of Results for Sequencing Problems with Controllable Processing Times, *Discrete Appl. Math.* **26**, 271 – 287.

[11] E. Nowicki and S. Zdrzalka (1995), A Bicriterion Approach to Preemptive Scheduling of Parallel Machines with Controllable Job Processing Times, *Discrete Appl. Math.* **63**, 271 – 287.

[12] N. V. Shakhlevich and V. A. Strusevich (2005), Preemptive Scheduling Problems with Controllable Processing Times, *Journal of Scheduling* **8**, 233 – 253.

[13] L. N. van Wassenhove and K. R. Baker (1982), A Bicriterion Approach to Time/Cost Tradeoffs in Sequencing, *Eur. J. Oper. Res.* **11**, 48 – 54.

[14] R. G. Vickson (1980a), Choosing the Job Sequence and Processing Times to Minimize Total Processing Plus Flow Cost on a Single Machine, *Oper. Res.* **28**, 1155 – 1167.

[15] R. G. Vickson (1980b), Two Single Machine Sequencing Problems Involving Controllable Job Processing Times, *AIIE Trans.* **12**, 258 – 262.

[16] G. Wan, B. P. C. Yen and C. L. Li (2001), Single Machine Scheduling to Minimize Total Compression Plus Weighted Flow Cost is NP-hard, *Inform. Process. Lett.* **79**, 273 – 280.

Computing Lower Bounds for the Schedule of a Multifunction Radar

Emilie Winter
Thales Systèmes Aéroportés, 2, av Gay-Lussac, 78851 Elancourt, France, emilie.winter@fr.thalesgroup.com
Ruslan Sadykov
Lix, Cnrs, École Polytechnique, 91128 Palaiseau, France, sadykov@lix.polytechnique.fr

Among several other tasks, the radar of a fighter has to search, track and identify potential targets. The waveforms used by the radar for each of these tasks are most often incompatible and hence, cannot be processed simultaneously. Moreover, these tasks are repeated several times in a cyclic fashion. Altogether, this defines a complex scheduling problem that impacts a lot on the quality of the radar's output. In [10], a formal framework has been defined for this real time scheduling problem and we have introduced several techniques to compute efficient schedules for the radar. In order to evalutate the quality of these schedules, we present in this paper, two algorithms that compute high lower bounds for the problem. The first one is based upon a column generation scheme, the other one consists in a relaxation of the time-indexed MIP formulation of the problem.

Keywords: Radar Scheduling, Column generation, Linear Programming.

1 Introduction

A radar is a system using radiowaves to detect the presence of objects in a given volume of space. It can also compute the range (distance) as well as the relative radial velocity of these objects. Airborne radars consist of a transmitter, a single antenna and a receiver. The transmitter generates radiowaves which are sent out in a narrow beam by the antenna in a specific direction. Objects located in the beam intercept this signal and scatter the energy in all directions. A portion of this energy is scattered back to the receiver of the radar listening to all potential echoes. See [8] and [9] for a detailed description of airborne radars.

Nowadays, most of the airborne radars have a mechanically steered antenna. With such antennas, the beam is perpendicular to the antenna which is pivoting so as to direct the beam. This paper is focused on recent radars with an Electronically Steered Antenna (ESA). An ESA is a planar array antenna made of many individual radiating elements. Unlike a mechanically steered antenna, an ESA lies in a fix position on the aircraft. The phase of the radiowaves is controlled electronically so that the radar beam lights up the desired direction.

One of the key advantages of ESA is that the beam is extremely agile. As it is not subjected to the mechanical inertia of the antenna, it can be moved instantaneously from one part of the space to another, even outside the search domain (Figure 1). Moreover, we consider that the radar can switch instantaneously to an appropriate waveform.

The following tasks have to be achieved by an airborne radar.

- **Research.** The radar is sweeping across a domain to detect the potential presence of targets inside.
- **Tracking.** The radar is closely monitoring the behavior of targets (initially detected at the Research stage).

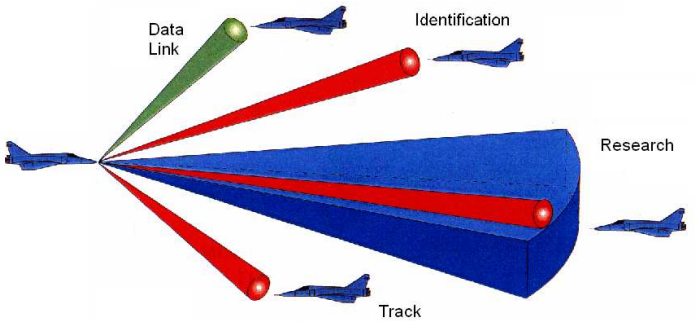


Figure 1: Functions of ESA Radars

- **Data Link.** The radar is used as a communication tool with other platforms.
- **Calibration.** The radar is performing cyclic calibration to ensure a high reliability level.

Research, Tracking, Data Link and Calibration require incompatible waveforms. So all corresponding tasks have to be *scheduled to ensure they do not overlap in time*. Moreover, Research, Tracking, Data Link and Calibration tasks have to be repeated in a more or less regular fashion. The execution of any single radar operation is called a *dwell*. Depending on the task, the periodicity constraint between dwells of a same type can be extremely important or not. For instance, as it is absolutely forbidden to loose a tracked target, tracking dwells have to be repeated with high regularity. Likewise, data link dwells are played at regular intervals, but there, the regularity constraint is much higher. Finally, to ensure a surveillance of great quality, in any circumstances, the radar has to play at least a minimum amount of research dwells.

It results from the use of only one device for all radar functions certain limitations. For instance, as different dwells cannot be played at the same time, while the radar is used in tracking mode, the pilot has no knowledge of the evolution of the tactical situation, that is new targets coming and so on. Also, the more targets to track, the less time for the search. Thus, the problem is how to use the radar time ressources to keep at best the whole situation awareness. We are interested with the scheduling of the dwells to make the radar more efficient while meeting the constraints unformally described above.

Barbaresco [1] describes a strongly related framework: Tasks are scheduled on a frame duration and are executed while the next schedule is computed. To schedule the tasks, Barbaresco associates deadlines to tasks and uses a heuristic called EDF (Earliest Deadline First). If one of the most important tasks is completed after its deadline then some tasks are removed of the system to reduce the load and the scheduling procedure is run again. The empirical study led in [1] shows that simple scheduling heuristics often improve the behavior of the radar.

The scheduling of the radar tasks can also be seen as a timing problem in which the tasks are scheduled to minimize an Earliness-Tardiness criterion [6].

Another scheduling problem for radars is the problem of interleaving tasks corresponding to receiving and sending data [7]. We study a situation where we do not have interleaving and both the sending and the receiving tasks are modeled as a unique task.

We have proposed in [10] a formal model based on cost functions that describes the problem. This model allows one to associate a cost, *i.e.* a score, to the schedules and it has been used to compare several scheduling heuristics. It has also been used to compute lower bounds to estimate the quality of our solutions. This paper focuses on the search of another lower bound by column

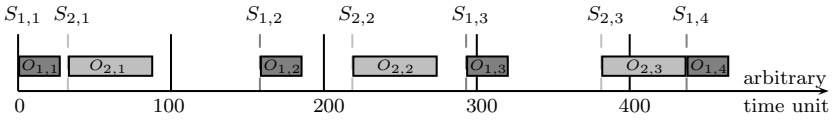


Figure 2: Scheduling Constraints

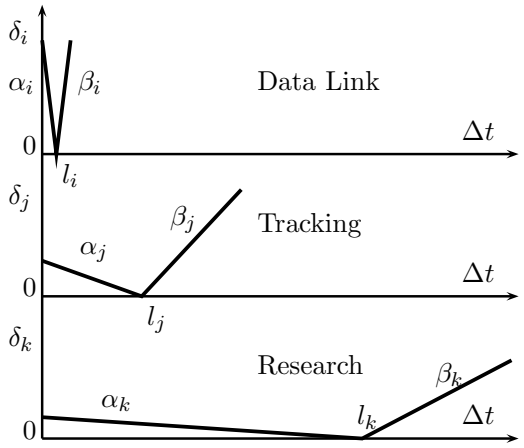


Figure 3: Typical Cost Functions for different jobs

generation. First, we re-introduce the model used in [10], adding some new notations. Then, we present an ILP formulation for the problem and we explain how to solve it by a column generation method. Further, we recall how we have computed time-indexed lower bounds in [10] and finally, we compare our experimental results to the column generation method.

2 Notations

We define a "*job*" as a single radar function that must be repeated in a more or less strict manner. A job is either the track of a given target or a data link or the "research". As several dwells are required to "execute" most jobs, a job consists of a set of non-preemptive "operations" which must obey a kind of periodicity rule. As shown on Figure 2, jobs are not necessarily executed with strict periodicity. The temporal gap between the two first operations of job 1 is greater than the gap between the operations 2 and 3. This schedule is feasible if performance constraints are ensured. The radar is seen as a single machine on which jobs have to be processed.

To model the problem, we associate cost functions δ_i to the distance between starting times of consecutive operations of the same job i. They are V-Shaped, *i.e.* $\delta_i(x) = \max(\alpha_i(l_i - x), \beta_i(x - l_i))$ where $l_i, \alpha_i \geq 0$ and $\beta_i \geq 0$ are respectively the ideal distance between two starting times and the penalty weights associated to a smaller (resp. larger) inter-distance. It derives from the penalty weights that some operations have greater priorities than others at a one time.

For each kind of job i, we have defined, in collaboration with radar engineers, cost functions features that fit real life scenario. An example of what cost functions can be is described Figure 3.

We are now ready to formally define the problem. Given n jobs 1, 2,..., n. Each job i is made of $n(i)$ consecutive and identical operations $O_{i1}, ..., O_{in(i)}$ with integer processing time p_i. For each

job i, we have a penalty function δ_i associated to the distance between starting times of consecutive operations. We are also given an integer $H \geq \sum_i n(i)p_i$ that represents the horizon of the schedule (*i.e.*, all operations have to be processed between 0 and H).

A set of starting times S_{ij} defines a feasible schedule if and only if (1) operations start after or at 0 and are not completed later than H and (2) operations do not overlap, *i.e.*, for any pair of operations $O_{ij}, O_{i'j'}$, $S_{ij}+p_i \leq S_{i'j'}$ or $S_{i'j'}+p_{i'} \leq S_{ij}$. The cost associated to the schedule is exactly the sum over all jobs i of $\sum_{u=1}^{n(i)-1} \delta_i(S_{iu+1} - S_{iu})$. The scheduling heuristics in [10] aim to compute the optimal schedule for the radar, that is the schedule that minimizes $\sum_{u=1}^{n(i)-1} \delta_i(S_{iu+1} - S_{iu})$.

Note that, in practice, the problem is solved continuously and the schedule followed in the past interacts with the schedule under construction. Stated another way, many jobs have been started a long time ago. This feature does not change the combinatorial structure of the problem and to keep things simple, we omit all details about it.

In order to compute lower bounds by column generation, we use some additional notations :

For each job i, we enumerate all the possible schedules. Sch_{ik} is the k^{th} possible schedule for job i, that is for i and k given, the starting times of all the operations of job i are known. The number of possible schedules for job i is denoted s_i and $s_i = \frac{Hr!}{n(i)!(Hr-n(i))!}$, where $Hr = H - p_i + 1$. The cost to play Sch_{ik} is $cost_{ik}$. $cost_{ik}$ can be computed thanks to the cost functions δ_i defined above. We also define a binary variable X_{ik} that equals 1 if the schedule Sch_{ik} is followed, 0 otherwise. Finally, we define the following function: $\delta(i,k,t)$ equals 1 if S_{ik} is active at time t, 0 otherwise.

3 ILP Formulation

The scheduling problem can be written as an integer linear program : ILP minimizes the summation of the costs of all the jobs in the sequence, making sure jobs do not overlap in time (1^{st} set of constraints) and that one and only one schedule is performed for each job i (2^{nd} set of constraints).

$$\text{(ILP)} : \min \sum_{i=1}^{n} \sum_{k=1}^{s_i} cost_{ik} X_{ik}$$

$$s.t. \begin{cases} \forall t, \sum_{i=1}^{n} \sum_{k=1}^{s_i} \delta(i,k,t) \cdot X_{ik} \leq 1 \\ \forall i, \sum_{k=1}^{s_i} X_{ik} = 1 \\ \forall i, \forall k, X_{ik} \in \{0,1\} \end{cases}$$

To compute lower bounds by column generation, we consider the relaxation $(ILPR)$ of (ILP). For convenience matters, equation (2) has been changed in an inequality *greater or equal than*, which is equivalent, as it is a minimization problem.

$$\text{(ILPR)} : \min \sum_{i=1}^{n} \sum_{k=1}^{s_i} cost_{ik} X_{ik}$$

$$s.t. \begin{cases} \forall t, -\sum_{i=1}^{n} \sum_{k=1}^{s_i} \delta(i,k,t) \cdot X_{ik} \geq -1 \\ \forall i, \sum_{k=1}^{s_i} X_{ik} \geq 1 \\ \forall i, \forall k, 0 \leq X_{ik} \leq 1 \end{cases}$$

To this point, we have stated the problem as a linear program with a great number of variables, $\sum_i^n s_i$. As the problem has many variables (columns) but relatively few constraints ($H+n$), column generation may be beneficial. The idea is that the linear program is too large to consider all the variables explicitly. Since most of the variables will be zero in the optimal solution, only a subset of variables needs to be considered when solving the problem. Column generation allows to generate only the variables that are required to decrease the objective function. *i.e.* to find variables with negative reduced cost. (see [4] for an introduction to column generation).

In order to characterize the optimal solution, we write the dual problem which is used to identify a new variable. This problem is known as the Pricing Problem and it is developed in the next section.

4 The Pricing Problem

Let Z_t and Y_i be respectively the dual constraints associated to the first and the second set of constraints of ILPR. The dual of ILPR is :

$$(DUAL) \ : \qquad \max \sum_{i=1}^{n} Y_i - \sum_{t=0}^{H-1} Z_t$$

$$s.t. \begin{cases} Y_i - \sum_{t=0}^{H-1} \delta(i,k,t) \cdot Z_t \leq cost_{ik} & 1 \leq i \leq n, \ 1 \leq k \leq ns_i \\ Z_t \geq 0 & 0 \leq t \leq H-1 \\ Y_i \geq 0 & 1 \leq i \leq n \end{cases}$$

In order to be practical, we must be able to solve the pricing problem. We obtain dual prices for each of the constraints of the relaxed problem ILPR. Given that set of dual values, if a constraint of DUAL is violated, then a variable with negative reduced cost has been identified in ILPR. This variable is added to ILPR. ILPR is solved again to generate a new set of dual values. The process is repeated until no negative reduced cost variables are identified. When all the constraints of DUAL are satisfied, the solution of ILPR is optimal.

Thus, we have some given (fixed) values for Y_i and Z_t and we are looking for a constraint

$$Y_i - \sum_{t=0}^{H-1} \delta(i,k,t) \cdot Z_t \leq cost_{ik}$$

that could be violated. Stated in scheduling terms, this means that, for all i, we are looking for a schedule of job i (*i.e.*, an index $k \leq s_i$) that minimizes

$$cost_{ik} + \sum_{t=0}^{H-1} \delta(i,k,t) \cdot Z_t. \tag{1}$$

We build such a schedule by dynamic programing. Let j be the number of operations of job i that still have to be scheduled. Let $P_i(j,t)$ denote the minimal cost of the schedule of the last $n(i)-j+1$ operations of job i assuming that the last $(n(i)-j+1)^{th}$ operation is completed at time t.

It is obvious that the first operation of job i to be scheduled has the minimum value of $\min_{0 \leq t \leq H} P_i(1,t)$. If the j^{th} operation is assumed to complete at time t, then operation $j+1$ must complete at some time t' that is more or equal to $t+p_i$. Let us also define $Z(i,t) = \sum_{\theta=t}^{t+p_i-1} Z_\theta$ and $F_i(t'-t)$ the cost introduced by the gap between two consecutive operations of job i. Now, we have

the following recursion that defines the minimum cost of the schedule such that the completion time of operation j

$$\begin{aligned} P_i(n(i),t) &= Z(i,t), \\ P_i(j,t) &= Z(i,t) + \min_{t+p_i \le t' \le H} \left\{ F_i(t'-t) + P_i(j+1,t') \right\}, \quad \forall j < n(i). \end{aligned}$$

Using the recursion above, this value can be found in $O(n(i) \cdot H^2)$ time, which could be improved. But there is a way to solve the pricing problem faster using the structure of the objective function F_i.

Consider again job i. Note that values $P_i(n(i),t)$ for all the time moments t can be computed in $O(H)$ time. Suppose now that we know the values $P_i(j+1,t)$, $j < n(i)$, $t \le H$. Then, we compute the following values for all time moments t.

$$\begin{aligned} a_i(j,t) &= \operatorname*{argmin}_{0 \le t' \le t} \left\{ \alpha_i(t-t') + P_i(j+1,t') \right\}, \\ b_i(j,t) &= \operatorname*{argmin}_{t \le t' \le H} \left\{ \beta_i(t'-t) + P_i(j+1,t') \right\}. \end{aligned}$$

This can be done in $O(H)$ time using the following recursion, $a_i(j,0) = 0$, $b_i(j,H) = H$, and

$$\begin{aligned} a_i(j,t) &= \begin{cases} t, & P_i(j+1,t) < P(j+1, a_i(j,t-1)) + \alpha_i \cdot (t - a_i(j,t-1)), \\ a_i(j,t-1), & \text{otherwise}, \end{cases} \\ b_i(j,t) &= \begin{cases} t, & P_i(j+1,t) < P(j+1, b_i(j,t+1)) + \beta_i \cdot (b_i(j,t+1) - t), \\ b_i(j,t+1), & \text{otherwise}. \end{cases} \end{aligned}$$

Let d_i' equal $\max\{d_i, p_i\}$. Using the values $a_i(j,t)$ and $b_i(j,t)$, the main recursion can be rewritten in the following way.

$$P_i(j,t) = Z(i,t) + \min \left\{ A, F_i(b_i(j,t+d_i') - t) + P_i(j+1, b_i(j,t+d_i')) \right\}, \tag{2}$$

$$\text{where} \quad A = \begin{cases} +\infty, & p_i \ge d_i, \\ F_i(a_i(j,t+d_i) - t) + P_i(j+1, a_i(j,t+d_i)), & p_i < d_i \text{ and } a_i(j,t+d_i) \ge t + p_i, \\ \min\limits_{t+p_i \le t' < t+d_i} \left\{ F_i(t'-t) + P_i(j+1,t') \right\}, & p_i < d_i \text{ and } a_i(j,t+d_i) < t + p_i. \end{cases}$$

Then, the theoretical complexity of the pricing problem for job i is reduced to $O\Big(n(i) \cdot H \cdot \max\{1, d_i - p_i\}\Big)$. Moreover, in practice, the resolution of the pricing problem for all the jobs is computable in $O(nH)$ time if one uses the updated recursion (2). Experiments have shown that the time needed to solve the pricing problem is negligible in comparison with the overall running time of the column generation algorithm.

For each job i, we check whether the minimum value of (1) is less than Y_i. If it is, we add to ILPR the column which corresponds to the schedule which minimises (1). If $\min_{k \le s_i}\{(1)\} \ge Y_i$ for all i, an optimal solution of ILPR is found. The value of this solution gives us a lower bound on the optimal solution of the radar problem.

5 Computing Lower Bounds by relaxing a Time-Indexed MIP

Let X_{ijt} be a binary variable which equals 1 if O_{ij} is played at time t, 0 otherwise. The optimal total minimum cost of the schedule can be computed by the Mixed Integer Linear Program below.

The first two constraints are due to the cost functions, The following one links the starting times to the X_{ijt} variables. The three next ones ensure respectively that each operation is scheduled at some time point, that precedence constraints between operations are met and finally that over all operations, only one can start at a time.

$$\min \sum_{i=1}^{n} \sum_{j=2}^{n(i)} W_{ij}$$

$$\begin{cases} W_{ij} \geq \alpha_i(l_i - S_{ij} + S_{ij-1}) & 1 \leq i \leq n, 2 \leq j \leq n(i) \\ W_{ij} \geq \beta_i(S_{ij} - S_{ij-1} - l_i) & 1 \leq i \leq n, 2 \leq j \leq n(i) \\ S_{ij} = \sum_{t=0}^{H-1} tX_{ijt} & 1 \leq i \leq n, 1 \leq j \leq n(i) \\ \sum_{t=0}^{H} X_{ijt} = 1 & 1 \leq i \leq n, 1 \leq j \leq n(i) \\ S_{ij} + p_i \leq S_{ij+1} & 1 \leq i \leq n, 1 \leq j \leq n(i) \\ \sum_{i=1}^{n} \sum_{j=2}^{n(i)} \sum_{t'=t-p_i+1}^{t} X_{ijt'} \leq 1 & 1 \leq i \leq n, 1 \leq j \leq n(i), 0 \leq t \leq H-1 \\ 0 \leq S_{ij} \leq H - p_i & 1 \leq i \leq n, 1 \leq j \leq n(i) \\ 0 \leq W_{ij} & 1 \leq i \leq n, 2 \leq j \leq n(i) \\ X_{ijt} \in \{0,1\}, \ integer & 1 \leq i \leq n, \ 1 \leq j \leq n_i, 0 \leq t \leq H-1 \\ S_{i0} < 0, \ \text{known} & 1 \leq i \leq n \end{cases}$$

The program gives the optimal solution but runs with prohibitive computation time Still, this program allows us to compute lower bounds by relaxing the integrity of the binary variable X_{ijt}, that is making the variables X_{ijt} real.

6 Experimental Results

Together with Thales engineering department, we have generated 25 instances that represent real life situations. The total number of operations varies from 44 to 96, the load of the radar varies from 45% to 100 % and the slopes of the cost functions α and β have been chosen to model different kind of situations. The local search and time-indexed experiments have run on DELL Latitude D600 laptop running Linux with the GNU Linear Programming Kit [2] as a LP solver while the column generation method has been run with Cplex [3]. For the local search, the run time has been limited to 1.5 second.

In any case, the column generation method has provided an upper lower bound than the time-indexed method. Let us consider the twelve instances up to 83 operations. We have obtained the same results with the local search and the column generation (respectively time-indexed) lower bound on 9 instances out of 12 (respectively, 2 out of 12). Thus, for those instances we know we have reached the optimal solution both with the scheduling heuristic and the lower bound. By looking at the thirteen instances with more than 91 operations, we have noticed that the local search doesn't run long enough to obtain as good results.

For a given instance, we denote by TI the value of the time-indexed lower bound and CG the column generation one. We have computed (TI/CG)*100 and (CG/LS)*100 for all the instances. The mean of the results in percentage appear in the comparative statement below.

Number of operations per instance	Number of instances	Mean of (TI/CG)*100 in %	Mean of (CG/LS)*100 in %
≤ 83	12	0.960	0.795
≥ 91	13	0.858	0.206

7 Conclusion

In this paper, we have proposed two methods to compute lower bounds for the radar scheduling problem. These approaches have been tested on a set of instances close from real life scenariosẆe have shown experimentally that the column generation algorithm provides stronger lower bounds than the linear programming relaxation of the time-indexed formulation of the problem. The lower bounds obtained have allowed us to evaluate the quality of the local search algorithm presented in [10] to schedule the tasks of the radar. It has been shown that it performs very well when the convergence time choosen for the algorithm fits the number of operations of the test instances. Thus, for a half of these instances, the optimality of the solutions found has been proved. However, for the test instances with more than 90 operations, the gap between lower and upper bounds known is still high. In the future, we hope to reduce this gap both by improving the local search algorithm and proposing methods to tighten lower bounds.

References

[1] F. Barbaresco (2003), Approche Cognitive de la Gestion Radar, *Proceedings of the conference Cogis 2003*.

[2] GNU Linear Programming Kit home page. http://www.gnu.org/software/glpk /glpk.html

[3] Cplex home page. http://www.ilog.com/products/cplex/

[4] G.B. Dantzig, P. Wolfe (1960), Decomposition principle for linear programs, *Oper. Res.* **8**, 101 – 111.

[5] M. Elshafei, H. D. Sherali, J. C. Smith (2003), Radar pulse interleaving for multi-target tracking, *Naval Research Logistics* **51**(1), 72 – 94.

[6] Y. Hendel (2005), Contributions à l'ordonnancement juste à temps, rapport de thèse, université Pierre et Marie Curie, laboratoire Lip6/SysDef.

[7] A. J. Orman, C. N. Potts (1997), On the Complexity of Coupled-task Scheduling, *Discrete Applied Mathematics* **72**(1-2), 141 – 154.

[8] M. I. Skolnik (1979), Introduction to Radar Systems, 2^{nd} edition, *McGraw-Hill Kogakusha*.

[9] G. W. Stimson (1998), *Introduction to Airborne Radar*, 2^{nd} Edition. SciTech Publishing, Inc.

[10] E. Winter, Ph. Baptiste (2007), On Scheduling a Multifunction Radar , accepted for publication in *Aerospace Science and Technologies*.

The Strength of Priority Algorithms

Yakov Zinder
University of Technology, Sydney, Department of Mathematics, PO Box 123, Broadway, NSW 2007, Australia,
yakov.zinder@uts.edu.au

The paper is concerned with scheduling problems with partially ordered unit execution time tasks and parallel identical machines. For the algorithms, constituting a broad class of algorithms for the maximum lateness problem, the paper introduces the notion of a strength, which characterizes the algorithm's worst-case performance. Using this formal framework, a positive answer is given to the question of the existence of a strongest algorithm, i.e. an algorithm with the best worst-case performance. The idea of this algorithm can be generalized to the maximum lateness problem with release times. The paper shows that this idea in combination with the idea of the Garey-Johnson algorithm leads to an exact algorithm for the maximum cost problem with release times and partially ordered unit execution time tasks.

Keywords: Machine Scheduling, Theoretical Scheduling

1 Scheduling Problems with UET Tasks

The problem of scheduling a partially ordered finite set of unit execution time (UET) tasks on parallel machines with the criterion of maximum lateness is one of the central in scheduling theory. This problem can be stated as follows. A finite set $N = \{1, \ldots, n\}$ of tasks (jobs, operations) is to be processed on $m > 1$ identical machines (processors) subject to precedence constraints in the form of an anti-reflexive, anti-symmetric and transitive binary relation α on N. If task i precedes task j in α, i.e. $(i, j) \in \alpha$, then the processing of i must be completed before the processing of j begins. The processing of tasks commences at time $t = 0$. Each machine can process only one task at a time. Once a machine begins processing a task, it continues until completion, i.e. no preemptions are allowed. Each task can be processed by any machine and requires one unit of time. A schedule σ is a mapping which assigns to each task j a positive integer – its completion time $C_j(\sigma)$. Each task has a due time (also often referred to as a due date) d_j. The goal is to find a schedule that minimizes the criterion of maximum lateness

$$L_{max}(\sigma) = \max_{j \in N}[C_j(\sigma) - d_j].$$

All due dates are arbitrary rational numbers.

In the standard three-field notation (see for example [2]), this problem is denoted by $P|prec, p_j = 1|L_{max}$, where P and $prec$ indicate parallel identical machines and the presence of precedence constraints, the term $p_j = 1$ specifies that the processing time p_j of each task j is one unit of time, and L_{max} denotes the criterion of maximum lateness. If all due times are equal to zero, the maximum lateness problem $P|prec, p_j = 1|L_{max}$ becomes the makespan problem $P|prec, p_j = 1|C_{max}$ with the criterion

$$C_{max}(\sigma) = \max_{j \in N} C_j(\sigma).$$

It is well known that $P|prec, p_j = 1|C_{max}$, and therefore $P|prec, p_j = 1|L_{max}$, is NP-hard in the strong sense [12], [9]. Moreover, the $P|prec, p_j = 1|C_{max}$ problem remains NP-hard even if the partially ordered sets of tasks are restricted to the bipartite graphs [15]. The $P|prec, p_j = 1|L_{max}$

problem remains NP-hard even if the partially ordered sets of tasks are restricted to the out-trees [3]. The NP-hardness results boosted the interest to the worst-case performance of various approximation algorithms. The majority of these algorithms were originally developed for some particular cases of the $P|prec, p_j = 1|L_{max}$ problem and are different implementations of the same approach. A class of algorithms, implementing a generalization of this approach, was introduced in [14]. Following [14], these algorithms will be referred to as priority algorithms.

In what follows, we will analyze the worst-case performance of priority algorithms, introducing the notion of a strength, and will give a positive answer to the question of the existence of a strongest priority algorithm (with the best performance guarantee). In answering this question we will consider not only currently known algorithms, but also "yet to be discovered" algorithms, i.e. all possible algorithms in the considered class.

A straightforward generalization of the $P|prec, p_j = 1|L_{max}$ problem is the problem with release times, where the processing of each task j can commence only after the nonnegative integer release time r_j. We will assume that $\min_{j \in N} r_j = 0$. In the three-field notation, this problem is denoted by $P|prec, p_j = 1, r_j|L_{max}$. In the problem with unit communication delays, denoted in the three-field notation by $P|prec, p_j = 1, r_j, c_{ij} = 1|L_{max}$, in any feasible schedule σ, for each task j, there is at most one task i such that $(i, j) \in \alpha$ and $C_i(\sigma) = C_j(\sigma) - 1$ and there is at most one task g such that $(j, g) \in \alpha$ and $C_g(\sigma) = C_j(\sigma) + 1$. It will be shown how the idea implemented in the strongest priority algorithm can be combined with the idea in [6] and [7] in order to obtain exact algorithms for $P|prec, p_j = 1, r_j| \max \varphi_j(C_j(\sigma))$ and $P|prec, p_j = 1, r_j, c_{ij} = 1| \max \varphi_j(C_j(\sigma))$, where all φ_j are nondecreasing functions.

2 Priority Algorithms

A generalization of the approach, implemented in [3], [5] and [16] for the $P|prec, p_j = 1|L_{max}$ problem, and in [1], [4], [8], [10], [11] for a particular case of the maximum lateness problem, the makespan problem $P|prec, p_j = 1|C_{max}$, was considered in [14]. Following [14], for an arbitrary instance $\mathcal{P}$ of the $P|prec, p_j = 1|L_{max}$ problem, the corresponding partially ordered set of tasks will be denoted by $(N(\mathcal{P}), \alpha(\mathcal{P}))$ and the corresponding sets of due times will be denoted by $\{d_j(\mathcal{P}) : j \in N(\mathcal{P})\}$. A subset $Q \subseteq N(\mathcal{P})$ is $\alpha(\mathcal{P})$-closed if, for any $j \in Q$ and any $g \in N(\mathcal{P})$, the relation $(j, g) \in \alpha(\mathcal{P})$ implies $g \in Q$. For any partially ordered set (M, δ) and any $Q \subset M$,

$$\delta_Q = \delta \cap (Q \times Q).$$

For any instance $\mathcal{P}$ of the $P|prec, p_j = 1|L_{max}$ problem and any $\alpha(\mathcal{P})$-closed subset $Q \subseteq N(\mathcal{P})$, $\mathcal{P}_Q$ is the instance of $P|prec, p_j = 1|L_{max}$ with the partially ordered set of tasks $(Q, \alpha(\mathcal{P})_Q)$ and the set of due times $\{d_j(\mathcal{P}_Q) : j \in Q\}$ such that $d_j(\mathcal{P}_Q) = d_j(\mathcal{P})$ for all $j \in Q$.

A priority algorithm for the $P|prec, p_j = 1|L_{max}$ problem is an algorithm that, for any instance $\mathcal{P}$ of this problem, first determines an anti-reflexive, anti-symmetric and transitive relation $\delta(\mathcal{P})$ on $N(\mathcal{P})$, satisfying

$$\delta(\mathcal{P}_Q) \subseteq \delta(\mathcal{P})_Q$$

for any $\alpha(\mathcal{P})$-closed subset $Q \subseteq N(\mathcal{P})$, and then applies the List Algorithm to a result of topological sorting of the partially ordered set $(N(\mathcal{P}), \delta(\mathcal{P}))$.

Although [14] is concerned with the $P|prec, p_j = 1|L_{max}$ problem, i.e. the problem where all the release times are equal to zero, for the purpose of Section 4 it is convenient to describe the List Algorithm, assuming that each task j can be processed only after the associated nonnegative integer release time r_j. In this case, without loss of generality assume that if $(i, j) \in \alpha$, then $r_i \leq r_j - 1$. The List Algorithm is an iterative procedure assigning the completion times in order

which is determined by a list of tasks. Initially this list contains all tasks. Once a task is assigned its completion time, it is deleted from the current list. Let $\mathcal{L}$ be the current list of tasks. For any task j, let $\mathcal{L}(j)$ be the position of this task in $\mathcal{L}$. For example, if $\mathcal{L} = (j_1, ..., j_u)$, then $\mathcal{L}(j_k) = k$. Let $|\mathcal{L}|$ be the number of tasks in the current list. Let $\bar{\sigma}$ be a schedule constructed by the List Algorithm. Schedule $\bar{\sigma}$ will be referred to as a list schedule.

List Algorithm

1. Set $t = \min_{j \in \mathcal{L}} r_j + 1 = 1$, $i = 1$ and $w = 0$.
2. Scan $\mathcal{L}$ from left to right, starting from the task in position i, and find the first task j such that $r_j < t$ and $C_i(\bar{\sigma}) < t$ for all i such that $(i, j) \in \alpha$. If such j does not exist, go to Step 4.
3. Set $C_j(\bar{\sigma}) = t$, $i = \mathcal{L}(j)$, $w = w + 1$, and eliminate j from the list. If $|\mathcal{L}| = 0$, then stop. If $w < m$ and $i \leq |\mathcal{L}|$, then go to Step 2.
4. Set $t = \min[t, \min_{j \in \mathcal{L}} r_j] + 1$, $w = 0$, $i = 1$. Go to Step 2.

The definition of a priority algorithm does not specify how the relation $\delta(\mathcal{P})$ is determined. For example, the definition does not exclude algorithms that establish $\delta(\mathcal{P})$ by enumerating all feasible schedules or some very large subset of the set of all feasible schedules. The relation $\delta(\mathcal{P})$ can be interpreted as an assignment of tasks' priorities, where task j is of higher priority than task g if $(j, g) \in \delta(\mathcal{P})$, and these two tasks are of the same priority if $(j, g) \notin \delta(\mathcal{P})$ and $(g, j) \notin \delta(\mathcal{P})$. The result of topological sorting $\mathcal{L} = (j_1, ..., j_n)$ has the following property: for any j_i and j_k, the relation $(j_i, j_k) \in \delta(\mathcal{P})$ implies the inequality $i < k$. Therefore, each time when several tasks compete for the assignment of a completion time, the List Algorithm chooses among all these tasks a task with the highest priority.

The analysis of the class of priority algorithms, presented in [14], is based of the notion of a domain, which was introduced in [14] as a key characteristic of a priority algorithm. On the intuitive level, the domain of an algorithm is a family of partially ordered sets of tasks for which this algorithm can solve the $P|prec, p_j = 1|L_{max}$ problem. Rigorous definitions can be found in [14].

3 Strength of Priority Algorithms

Since the general $P|prec, p_j = 1|L_{max}$ problem is NP-hard in the strong sense, various algorithms, developed for this problem, are characterized by their worst-case performance. Commonly the worst-case performance of any algorithm A for the $P|prec, p_j = 1|L_{max}$ problem is described in the form

$$L^{\mathcal{P}}_{max}(\sigma^A_{\mathcal{P}}) \leq \gamma L^{\mathcal{P}}_{max}(\sigma^*_{\mathcal{P}}) + \beta \max_{j \in N(\mathcal{P})} d_j(\mathcal{P}) - \eta, \quad (1)$$

where $\mathcal{P}$ is an arbitrary instance of the $P|prec, p_j = 1|L_{max}$ problem, $\sigma^A_{\mathcal{P}}$ is a schedule, constructed by A for the instance $\mathcal{P}$, $\sigma^*_{\mathcal{P}}$ is an optimal schedule for this instance,

$$L^{\mathcal{P}}_{max}(\sigma) = \max_{j \in N(\mathcal{P})} \{C_j(\sigma) - d_j(\mathcal{P})\},$$

and γ, β and η are some constants, which remain the same for all instances $\mathcal{P}$ of the $P|prec, p_j = 1|L_{max}$ problem.

Lemma 1 *For any priority algorithm A there exists a constant γ such that*

$$L^{\mathcal{P}}_{max}(\sigma^A_{\mathcal{P}}) \leq \gamma L^{\mathcal{P}}_{max}(\sigma^*_{\mathcal{P}}) + (\gamma - 1) \max_{j \in N(\mathcal{P})} d_j(\mathcal{P}) \tag{2}$$

for all instances $\mathcal{P}$ of the $P|prec, p_j = 1|L_{max}$ problem.

The series composition of partially ordered sets (N_1, α_1) and (N_2, α_2) is the partially ordered set $(N_1 \cup N_2, \alpha_1 \cup \alpha_2 \cup (N_1 \times N_2))$. Lemma 1 leads to the following definition. For any priority algorithm A, which domain is not empty and is closed with respect of the series composition, its strength, denoted by $\gamma(A)$, is the smallest γ, satisfying the inequality (2) for all instances $\mathcal{P}$ of the $P|prec, p_j = 1|L_{max}$ problem. Consequently, priority algorithm A with strength $\gamma(A)$ is stronger than priority algorithm B with strength $\gamma(B)$ if $\gamma(A) \leq \gamma(B)$.

The above definitions raise the following questions:

(a) Does there exist the strongest priority algorithm?

(b) If the strongest priority algorithm exists, what is its strength?

(c) If the strongest priority algorithm exists, what is its computational complexity?

The following theorem answers these questions.

Theorem 1 *The polynomial-time algorithm, presented in [16], is the strongest priority algorithm.*

As has been shown in [16], the polynomial-time algorithm, presented in this paper, satisfies the following performance guarantee

$$L^{\mathcal{P}}_{max}(\sigma^A_{\mathcal{P}}) \leq \left(2 - \frac{2}{m}\right) L^{\mathcal{P}}_{max}(\sigma^*_{\mathcal{P}}) + \left(1 - \frac{2}{m}\right) \max_{j \in N(\mathcal{P})} d_j(\mathcal{P}) - r(m), \tag{3}$$

where $\sigma^A_{\mathcal{P}}$ is a schedule constructed by this algorithm, $\sigma^*_{\mathcal{P}}$ is an optimal schedule for the maximum lateness problem, and

$$r(m) = \begin{cases} \dfrac{m-3}{m} & \text{for } m \text{ odd} \\ \dfrac{m-2}{m} & \text{for } m \text{ even.} \end{cases} \tag{4}$$

4 Minimization of Maximum Cost

The results of [14] and Theorem 1 characterize the capability of priority algorithms and motivate the exploration of other approaches. One of the well known algorithms, which does not satisfy the above definition of a apriority algorithm, is the classical Garey-Johnson algorithm [6]. Originally, this algorithm was intended for the $P2|prec, p_j = 1, r_j|L_{max}$ problem, i.e. the problem with only two machines, but allows a generalization for the case of an arbitrary number of machines [7].

In order to apply this generalization, which following [7] will be referred to as a GJ-algorithm, the original $P|prec, p_j = 1|L_{max}$ problem should be transformed into the $P|prec, p_j = 1, r_j|L_{max}$ problem. For a partially ordered set (N, α), k-path from any $j \in N$ to any $g \in N$ is a sequence j_0, ..., j_k such that $j_0 = j$, $j_k = g$, and for any $0 \leq i \leq k-1$, $(j_i, j_{i+1}) \in \alpha$. If $(j, g) \in \alpha$, then let $l(j, g)$ be the largest k for which there exists a k-path from j to g. If $(j, g) \notin \alpha$, then let $l(j, g) = 0$. Then $P|prec, p_j = 1|L_{max}$ can be transformed into $P|prec, p_j = 1, r_j|L_{max}$ by introducing for each $j \in N$ the release time

$$r_j = \max_{i \in N} l(i, j).$$

As has been proven in [7] for any instance of the $P|prec, p_j = 1, r_j|L_{max}$ problem

$$L_{max}(\sigma^{GJ}) \leq [2 - \alpha(m)]L_{max}(\sigma^*) + [1 - \alpha(m)] \max_{v \in N} d_v - [1 - \alpha(m)],$$

where σ^{GJ} is a schedule constructed by the GJ-algorithm, σ^* is an optimal schedule, and

$$\alpha(m) = \begin{cases} \dfrac{2}{m+1} & \text{if} \quad m \quad \text{is odd} \\ \dfrac{2}{m} & \text{if} \quad m \quad \text{is even} \end{cases}.$$

Another algorithm for the $P|prec, p_j = 1, r_j|L_{max}$ problem was presented in [13] and has performance guarantee (3)-(4). This algorithm can be considered as a generalization of [16]. The ideas, presented in [6], [7] and [13], lead to the algorithm below, constructing an exact solution to the $P|prec, p_j = 1, r_j| \max \varphi_j(C_j)$ problem, where all φ_j are nondecreasing functions. Without loss of generality, we again assume that if $(i, j) \in \alpha$, then $r_j \geq r_i + 1$. We also assume that for any list schedule σ, $C_{max}(\sigma) \leq n$. If this condition does not hold, then the problem can be decomposed into two separate problems.

The algorithm below is based on the idea of Δ-modified due dates, which according to [7] (see also [6]) are defined as follows. Let D_1, ..., D_n be arbitrary nonnegative integers, then for any task i and any two numbers s and h such that

$$r_i \leq s \leq D_i \leq h, \tag{5}$$

$S(i, s, h)$ denotes the set of all tasks j such that $j \neq i$, $D_j \leq h$, and either $(i, j) \in \alpha$ or $r_j \geq s$. Integers $D_1, \ldots, D_n$ are consistent if for every task j

$$r_j \leq D_j - 1,$$

and for any task i and any two integers s and h, satisfying (5), either $D_i = s$ and $|S(i, s, h)| = m(h - s)$, or $|S(i, s, h)| < m(h - s)$. For any integer Δ, integers D_1, ..., D_n are Δ-modified due dates with respect to d_1, ..., d_n, if they are consistent and $D_i \leq d_i + \Delta$ for all $i \in N$.

For any set of due dates $d = \{d_1, ..., d_n\}$ let $\Delta(d)$ be the smallest Δ for which there exist Δ-modified due dates with respect to d_1, ..., d_n. As has been shown in [7], for any schedule σ,

$$\Delta(d) \leq \max_{j \in N}[C_j(\sigma) - d_j].$$

For any set of consistent due dates $x = \{x_1, ..., x_n\}$, σ^x will denote a list schedule, corresponding to a list where tasks are arranged in a nondecreasing order of $\{x_1, ..., x_n\}$. For any $i \in N$, $K(i)$ will denote the set of all tasks j such that $(i, j) \in \alpha$. For any positive integer t the time interval $[t - 1, t]$ will be referred to as a time slot t.

Algorithm

1. Set $V = \max_{j \in N} \varphi_j(r_j + 1)$.

2. For each $j \in N$, find d_j: the largest τ among all integers τ such that $\tau \leq n$ and $\varphi_j(\tau) \leq V$. For the set $d = \{d_1, ..., d_n\}$, find $\Delta(d)$ and the corresponding set of $\Delta(d)$-modified due dates, say $z = \{z_1, ..., z_n\}$. Set $\delta = \Delta(d)$. If $\Delta(d) = 0$, then go to Step 4.

3. Set
$$V = \min_{\{j:\, d_j+\delta\le n\}} \varphi_j(d_j+\delta)$$
and go to Step 2.

4. If $L(\sigma^z)=0$, then stop: σ^z is an optimal schedule. Otherwise, set $D=\{z\}$ and $\delta = n$.

5. If $D=\emptyset$, then go to Step 3. Otherwise, among all sets of consistent due dates $y=\{y_1,...,y_n\}\in D$ select a set, say $x=\{x_1,...,x_n\}$, with the smallest value of
$$\max_{j\in N}[C_j(\sigma^y)-y_j].$$
Eliminate x from D.

6. Among all tasks $j\in N$, satisfying the equality
$$C_j(\sigma^x)-x_j = \max_{j\in N}[C_j(\sigma^x)-x_j],$$
select one with the smallest $C_j(\sigma^x)$, say task g. Select T: the largest t among all integers $t<C_i(\sigma^x)$ such that the number of tasks with the completion time t is less than m or there exists j with $C_j(\sigma^x)=t$ and $x_j>x_g$.

7. If $r_g<T$, then set
$$U=\{j:\ T<C_j(\sigma^x)<C_g(\sigma^x) \text{ and } r_j<T\}.$$
Otherwise, set
$$U=\{j:\ T<C_j(\sigma^x)<C_g(\sigma^x) \text{ and } r_j<T\}\cup\{g\}.$$
Set
$$J=\{j:\ C_j(\sigma^x)=T \text{ and } K(j)\cap U\neq\emptyset\}.$$

8. Select an arbitrary $j\in J$ and find $\Delta(y)$ and the corresponding set of $\Delta(y)$-modified due dates, say $w=\{w_1,...,w_n\}$, for $y=\{y_1,...,y_n\}$ such that
$$y_i=\begin{cases} x_i, & \text{if } i\neq j\\ x_j=T-C_g(\sigma^x)+x_g, & \text{if } i=j\end{cases}$$
If $\Delta(y)>0$, then set $\delta=\min[\delta,\Delta(y)]$ and to Step 10.

9. If $\max_{j\in N}\varphi_j(C_j(\sigma^w))=V$, then stop: σ^w is an optimal schedule. Otherwise, set $D=D\cup\{w\}$.

10. If $J\neq\emptyset$, then go to Step 8. Otherwise, go to Step 5 .

The presented approach is also applicable to $P|prec,p_j=1,r_j,c_{ij}=1|\max\varphi_j(C_j(\sigma))$. The particular implementation of this approach can vary depending on cost functions φ_j. First computational experiments with an algorithm for the $P|prec,p_j=1|C_{max}$ problem showed its high effectiveness [17].

References

[1] B. Braschi and D. Trystram (1994), A new insight into the Coffman-Graham algorithm , *SIAM Journal on Computing* **2**3, 662 – 669.

[2] P. Brucker (2001), *Scheduling algorithms*, third edition, Springer, Berlin.

[3] P. Brucker, M.R. Garey and D.S. Johnson (1977), Scheduling equal-length tasks under tree-like precedence constraints to minimise maximum lateness , *Math. Oper. Res.* **2**, 275 – 284.

[4] E.G. Coffman, Jr and R.L. Graham (1972), Optimal scheduling for two-processor systems , *Acta Inform.* **1**, 200 – 213.

[5] M.R. Garey and D.S. Johnson (1976), Scheduling tasks with nonuniform deadlines on two processors , *Journal of the Association for Computing Machinery* **2**3, 461 – 467.

[6] M. R. Garey and D.S. Johnson (1977), Two-processor scheduling with start-time and deadlines, *SIAM Journal on Computing* **6**, 416 – 426.

[7] C. Hanen and Y. Zinder, The worst-case analysis of the Garey-Johnson algorithm, *submitted for publication*

[8] T.C. Hu (1961), Parallel sequencing and assembly line problems , *Operation Research* **9**, 841 – 848.

[9] J.K. Lenstra and A.H.G. Rinnooy Kan (1978), Complexity of scheduling under precedence constraints , *Oper. Res.* **2**6, 22 – 35.

[10] A. Moukrim (1999), Optimal scheduling on parallel machines for a new order class , *Operation research letters* **2**4, 91 – 95.

[11] C.H. Papadimitriou and M. Yannakakis (1979), Scheduling interval orderred tasks , *SIAM J. Comput.* **8**, 405 – 409.

[12] J.D. Ullman (1975), NP-complete scheduling problems , *J. Comput. System Sci.* **1**0, 384 – 393.

[13] Y. Zinder, An iterative algorithm for scheduling UET tasks with due dates and release times, *European Journal of Operational Research* **1**49, 404 – 416

[14] Y. Zinder and V.H. Do, Priority algorithms for scheduling UET tasks: domains and dominance, *submitted for publication*

[15] Y. Zinder and D. Roper (1995), A minimax combinatorial optimisation problem on an acyclic directed graph: polynomial-time algorithms and complexity , In: *Proceedings of the A.C. Aitken Centenary Conference*, Dunedin, 391 – 400.

[16] Y. Zinder and D. Roper (1998), An iterative algorithm for scheduling unit-time operations with precedence constraints to minimise the maximum lateness, *Annals of Operations Research* **8**1, 321 – 340.

[17] Y. Zinder, G. Singh and R. Weiskircher (2006), A new method of scheduling UET tasks on parallel machines, *Lecture Notes in Engineering and Computer Science*, Hong Kong, 796 – 801

Two-machine Shop Floor Scheduling Problem with Multi-Predecessor Constraints*

Xiandong Zhang
Department of Management Science, School of Management, Fudan University,
Shanghai 200433, China, xiandongzhang@fudan.edu.cn
Jatinder N.D. Gupta
College of Administrative Science, The University of Alabama in Huntsville,
Huntsville, AL 35899, USA, guptaj@uah.edu

This paper deals with a two-machine shop floor scheduling problem with multi-predecessor constraints. The problem is proved to be NP-hard in strong sense. The paper proves that under some additional constraints the Algorithm ISPT (Instant Shortest Processing Time first) results an optimal solution. A branch and bound algorithm is developed for the general case and some dominant rules are provided.

Keywords: Shop Floor Scheduling, Multi-predecessor Constraints, Complexity of Scheduling Problems, Branch and bound algorithm

1. Introduction

This paper addresses a two-machine shop floor scheduling problem with multi-predecessor constraints. We are given two machines A and B, and a set $J=\{J_1,J_2,...,J_n\}$ of jobs. Each job $J_j \in J$ must be first processed on machine A and then on machine B, therefore it is a kind of flow shop. The operation times are p_{1j} and p_{2j}, respectively, where $p_{1j}+p_{2j}>0$. Neither of the machines can process more than one job at a time. The objective is to find a schedule that minimizes the makespan C_{max}, which is the completion time of all jobs on both machines. This original problem is denoted by $F2||C_{max}$.

The $F2||C_{max}$ problem can be solved by Johnson's rule very conveniently. But considering multi-predecessor constraints, the problem may be difficult. The multi-predecessor constraints means job J_j may have multiple predecessors, and all the predecessors should be finished first in machine A before job J_j could be started in machine B, i.e. the operation of p_{2j} in machine B could not be started until the operations of p_{1j} and p_{1k}, $J_k \in J^k$, in machine A are finished, where J^k is the set of jobs that precede job J_j.

We write $J_k \xrightarrow{m} J_j$ and say that job J_k is one of the multi-predecessors of job J_j. Job J_j may have more than one predecessors. This type of constraint is defined as *multi-predecessor constraint*. The problem discussed in this paper is denoted by $F2|multi\text{-}predecessor|C_{max}$. Since a numerical example to illustrate where a permutation schedule is not optimal can be easily constructed, we need to find an optimal non-permutation schedule for the problem.

* Supported by China National Nature Science Foundation. Project No.: 70432001.

Traditionally, there are two types of precedence constraints [1]. For two jobs J_j and J_k, we write $J_j \xrightarrow{1} J_k$ and say that job J_j precedes job J_k if and only if job J_k cannot start on any machine until job J_j is completed on all machines it has to be processed on. The relation $\xrightarrow{1}$ will be called the *precedence relation of the first type*. On the other hand, for two jobs J_j and J_k, we write $J_j \xrightarrow{2} J_k$ and say that job J_j precedes job J_k if and only if job J_k cannot start on any machine until job J_j is completed on the same machine. The relation $\xrightarrow{2}$ will be called the *precedence relation of the second type*.

We write "*prec1*" or "*prec2*" in the second position of the standard notation for scheduling problems to indicate that there are precedence constraints either of the first or of the second, respectively. The F2|prec1|C_{max} problem is trivial, and the optimal makespan is the sum of all the operations time. The F2|prec2|C_{max} problem is NP-hard in the strong sense due to Strusevich [1] and Monma [2].

If $J_j \xrightarrow{2} J_k$, the operation of p_{1j} will precede the operation of p_{1k} and operation of p_{2j} will precede the operation of p_{2k}. Therefore, $J_j \xrightarrow{2} J_k$ implies $J_j \xrightarrow{m} J_k$, but the opposite is not true. The multi-predecessor condition is a kind of relaxation of *precedence relation of the second type*.

2. The Complexity of F2|multi-predecessor|C_{max}

In this part, we will prove that F2|multi-predecessor|C_{max} is strongly NP-hard. The proof is done by reducing F2|prec2|C_{max} to F2|multi-predecessor|C_{max}. We first show that an instance of F2|prec2|C_{max} could be transformed to an instance of F2|multi-predecessor|C_{max}. Then we prove that an optimal schedule of the instance of F2|multi-predecessor|C_{max} can be used to construct an optimal schedule of the instance of F2|prec2|C_{max}.

First, we need the following definition. For the F2|prec2|C_{max} problem, if $J_i \xrightarrow{2} J_j$ and $J_j \xrightarrow{2} J_k$, then $J_i \xrightarrow{2} J_k$. If $J_j \xrightarrow{2} J_k$ and there is no job J_i such that $J_j \xrightarrow{2} J_i \xrightarrow{2} J_k$, then job J_j is said to *directly precede* job J_k, and this is denoted by $J_j \xrightarrow{d2} J_k$.

Then, we use the following algorithm to transform an instance of F2|prec2|C_{max} to an instance of F2|multi-predecessor|C_{max}.

Algorithm T

Step 1 List all the job pairs with the *precedence relation of the second type,* including *directly preceded* job pairs and *indirectly preceded* job pairs in the instance of F2|prec2|C_{max}.

Step 2 Change all the *direct and indirect precedence relations of the second type* to the *multi-predecessor constraints.*

Because there are less than n^2 job pairs with precedence relations of the second type, the *algorithm T* is obviously polynomial. We call the original instance as the problem P and the transformed instance as the problem P'. We claim that a feasible schedule S' of problem P' can be modified to a feasible schedule S of problem P without increasing the makespan. The

following *algorithm M* describes the details of the modification. In the *algorithm M*, we will first change the order of operations in machine *B*, and then change the order of operations in machine *A*.

Algorithm M

(Part I. Change the order of operations in machine B.)

Step 1 Let $V = \emptyset$. Define a jobs set U which contains jobs without successors in *precedence relations of the second type*. For each job J_k in set U, find all the direct predecessors of the second type $J_j \square J$. If there is a contradiction in machine B that operation p_{2k} is scheduled before p_{2j}, then relocate operation p_{2j} just before p_{2k}.

Step 2 Let $V = V \bigcup U$. If $V = J$, go to Step 4. Otherwise, redefine the jobs set U which contains jobs having direct successors in set V. For each job J_k in set U, find all the direct predecessors of the second type $J_j \square J$. If there is a contradiction in machine B that operation p_{2k} is scheduled before p_{2j}, then relocate operation p_{2j} just before p_{2k}.

Step 3 Go to Step 2.

(Part II. Change the order of operations in machine *A*.)

Step 4 Let $N = \emptyset$. Define a jobs set M which contains jobs without predecessors in *precedence relations of the second type*. For each job J_j in set M, find all the direct successors of the second type $J_k \square J$. If there is a contradiction in machine A that operation p_{1k} is scheduled before p_{1j}, then relocate operation p_{1k} just after p_{1j}.

Step 5 Let $N = N \bigcup M$. If $N = J$, stop. Otherwise, redefine the jobs set M which contains jobs having direct predecessors in set N. For each job J_j in set M, find all the direct successors of the second type $J_k \square J$. If there is a contradiction in machine A that operation p_{1k} is scheduled before p_{1j}, then relocate operation p_{1k} just after p_{1j}.

Step 6 Go to Step 4.

Because there are at most n^2 job pairs with precedence relations of the second type, the *algorithm M* is also polynomial.

Theorem 2.1 *the F2|multi-predecessor|C_{max} problem is NP-hard in strong sense.*

Proof. The proof is done by reducing F2|prec2|C_{max} to F2|multi-predecessor|C_{max}. According to *Algorithm T*, an instance of F2|prec2|C_{max} could be transferred to an instance of F2|multi-predecessor|C_{max}. If an optimal schedule S^* could be found for the transformed instance of F2|multi-predecessor|C_{max}, we can use *Algorithm M* to change the schedule to a feasible schedule S^* for the original instance of F2|prec2|C_{max}. It is easy to verify that the modification in *Algorithm M* will not increase the makespan. Since F2|multi-predecessor|C_{max} contains less constraints than F2|prec2|C_{max}, the optimal makespan of problem P' is less than or equal to the optimal makespan of problem P. Therefore the modified schedule S^* of problem P' is an optimal schedule of problem P. With the strongly NP-hard property of F2|prec2|C_{max}, we know that the F2|multi-predecessor|C_{max} problem is strongly NP-hard. □

3. A branch and bound approach

Since the problem of F2|multi-predecessor|Cmax is NP-hard in strong sense, we will provide a

branch and bound algorithm in this paper. Before we describe the algorithm, some dominant rules will be developed.

Define Φ_j as an operations set which contains operations in machine A that have to be processed before the operation of p_{2j}. The property of set Φ_j can be described as following.

i) $\Phi_j \neq \emptyset$, because p_{1j} is one of the elements;

ii) Φ_j and Φ_k have intersection if an operation of p_{1s} precedes both the operations of p_{2j} and p_{2k}.

Define φ_j as a subset of Φ_j. φ_j can be the empty set $\emptyset$ or the whole *set* Φ_j.

Lemma 3.1 *Consider an optimal sequence in machine B. Without loss of generality, we change the subscripts of operations in the optimal sequence such that the sequence is* $p_{21}, \ldots, p_{2n}$. *Then the optimal sequence in machine A is in the order of* $\varphi_1, \varphi_2, \ldots, \varphi_n$.

Proof. The proof is done by induction. First, let $j = 1$. Consider operation p_{21}. Since all the operations in Φ_1 have to be finished before p_{21}, we can postpone all the operations not in Φ_1 until all the operations in Φ_1 are finished. Obviously, this will not increase the makespan. Therefore, φ_1 is equal to Φ_1. Then, let's consider operation p_{2j}. Take the order of $\varphi_1, \varphi_2, \ldots, \varphi_{j-1}$ as fixed in machine A. Since all the operations in Φ_j have to be finished before p_{2j}, and some operations in Φ_j have been finished, we can postpone all the rest operations not in Φ_j until all the operations in Φ_j are finished. This will not increase the makespan. □

From above lemma, we can have a corresponding sets of $\varphi_1, \varphi_2, \ldots, \varphi_n$ in machine A, if we have a operations sequence s in machine B, $p_{21}, \ldots, p_{2j}, \ldots, p_{2n}$.

Define $\Phi_j(k)$ as a subset of Φ_j. $\Phi_j(k)$ contains the all the operations in Φ_j which are not included in $\varphi_1, \varphi_2, \ldots, \varphi_{k-1}$. That is to say, if we put job j in the kth position in the sequence, $\Phi_j(k)$ contains all the unprocessed operations in machine A which precede p_{2j}. Clearly, we have $\Phi_j(1) = \Phi_j$.

Define A_j as the total operation time of Φ_j. We have

$$A_j = \sum_{p_{1i} \in \Phi_j} p_{1i}$$

Similarly define $A_j(k)$ as the total operation time of $\Phi_j(k)$.

$$A_j(k) = \sum_{p_{1i} \in \Phi_j(k)} p_{1i}$$

Clearly, $A_j(1) = A_j$.

If $A_j \leq p_{2j}$ for each job J_j in set J, we can form a sequence with the following *Instant Shortest Process Time* (*ISPT*) algorithm.

Algorithm ISPT

Step 1 Let k=1. Find a job with the smallest A_j in machine A. Ties may be broke arbitrarily. Assign this job as the first job in sequence s.

Step 2 Let $k=k$+1. Find the job J_j with the smallest $A_j(k)$. Ties may be broke arbitrarily. Assign job j as the kth job in sequence s.

Step 3 if $k=n$, Stop; else go to ***Step2***.

Theorem 3.1 *The sequence s generated from Algorithm ISPT is an optimal sequence for F2|multi-predecessor|Cmax if* $A_j \leq p_{2j}$ *for each job* J_j *in set J.*

Proof. From *Lemma 3.1*, we know that the operations sequence in machine A is decided by the operations sequence in machine B, and it is easy to verify that the operations sequences within φ_j, j=1,…,n, have no effect on the makespan.

The further proof is by contradiction. Suppose a schedule with contradiction is optimal. In this optimal schedule, there must be a pair of adjacent jobs, say job J_j followed by job J_k, that satisfy the condition of $A_j(l)>A_k(l)$, l is job J_j's position in the sequence.

It is suffices to show that the makespan is reduced after a pairwise interchange of jobs J_j and J_k in machine B.

Let Θ_{jk} be the intersection of $\Phi_j(l)$ and $\Phi_k(l)$. Because operations in Θ_{jk} are the predecessors for both operations of p_{2j} and p_{2k}, these operations can be finished firstly no matter the order of jobs j and k. Let $\Phi'_j(l)$ and $\Phi'_k(l)$ be the subsets of $\Phi_j(l)$ and $\Phi_k(l)$ without Θ_{jk}, respectively. Let $A'_j(l)$ and $A'_k(l)$ be the operation time of $\Phi'_j(l)$ and $\Phi'_k(l)$. Clearly, we have $A'_j(l) > A'_k(l)$. Since $\Phi'_j(l)$ and $\Phi'_k(l)$ have no intersections, and the operations in $\Phi'_j(l)$ have to be finished before p_{2j} and the operations in $\Phi'_k(l)$ have to be finished before p_{2k}, the shortest makespan for these two jobs can be achieved easily by *Johnson's rule*. If $A'_j(l) > A'_k(l)$, $A'_j(l) \leq A_j \leq p_{2j}$, and $A'_k(l) \leq A_k \leq p_{2k}$, the sequence of job J_k followed by job J_j will ensure the shortest finish time in machine B for jobs J_j and J_k. □

Corollary 3.1 *In the optimal sequence of a problem of F2|multi-predecessor|*C_{max}*, the order of jobs which have* $A_j \leq p_{2j}$*, follows the ISPT rule.*

Proof. The proof is by contradiction. Suppose another type of schedule is optimal. In this optimal schedule, there must be a pair of jobs, say job J_j followed (may not be immediately followed) by job J_k, that satisfy the condition of $A_j(p)>A_k(p)$, which have $A_j(p) \leq p_{2j}$, $A_k(p) \leq p_{2k}$, p is job J_j's position in the sequence. This can be illustrated by Fig. 3.1.

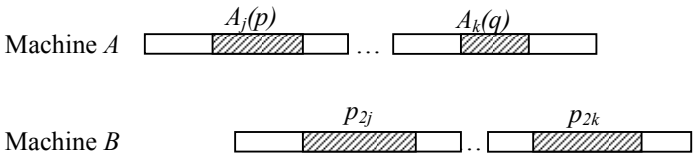


Fig. 3.1 Illustration for Corollary 3.1

It is easy to verify that if we relocate job J_k to the front of job J_j in machine B, and make the corresponding change in machine A based on *Lemma 3.1*, the total makespan will not be increased. □

Corollary 3.2 *In the optimal sequence of a problem of F2|multi-predecessor|Cmax, the order of jobs which have condition of $p_{1j} > p_{2j}$ and have no successors will follow the LPT rule in machine B, and all these kind of jobs will be sequenced after the jobs which have $A_j \le p_{2j}$.*

The proof is by contradiction and similar to the proofs in Theorem 3.1 and Corollary 3.1. □

The above two corollaries will help us in finding a better branching strategy. To find the bound for the original problem, we need following Lemma.

Lemma 3.2 The *Johnson's rule* is the optimal solution for the problem of F2||C_{max}. □

A branch and bound procedure for F2|multi-predecessor|C_{max} can be constructed as follows. First, based on *Lemma 3.1*, the sequence in machine *A* is decided by the sequence in machine *B*. So, we only need to develop the sequence in machine *B*. The branching operation may be based on the fact that schedules are developed starting from the beginning of the schedule. There is a single node at level *0* that is the top of the branching tree. At this node, none of the jobs has been put into any position in the sequence.

The branching step In branching level k ($k \ge 1$), there are k-1 jobs that have been scheduled. Define J^c as the set of jobs that are not scheduled, J^d as the subset of J^c which contains jobs of $A_j(1) \le p_{2j}$, J^e as another subset of J^c which contains jobs of $p_{1j} > p_{2j}$ and have no successors, and J^b as the complementary set of J^d and J^e in set J^c. If $J^d \neq \emptyset$, find the job J_p in J^d with the smallest $A_p(k)$, and add job J_p to J^b. If $J^d = \emptyset$, and $J^e \neq \emptyset$, find the job J_q in J^e with the largest p_{2q} and add job J_q to J^b. Every job in J^b can be put in the position k in the sequence. The number of jobs in J^b is the number of branches in level k. If $J^b = \emptyset$, we already get a complete sequence; otherwise, we get a partial sequence.

The bounding step Based on each newly generated partial sequence, consecutively append all unscheduled jobs into the partial sequence in the order of *Johnson's rule*, and compute their completion times without considering their precedence constraints. The completion time at machine *B* for the latest appended job is the lower bound for the makespan of this partial sequence. In level k, the partial sequence with the lowest lower bound will be selected as the next branching node.

The fathoming step At the beginning of the algorithm, the current optimal makespan Z^* is set at infinity, that is, $Z^*=\infty$. A partial sequence will be fathomed if either

(i) Its lower bound is not smaller than Z^*, or

(ii) It is a complete sequence.

If a partial sequence is fathomed by (ii), update Z^* with its associated lower bound whenever the latter has a smaller value.

Recursively implementing the branch-and-bound algorithm above will generate an optimal job sequence.

4. Conclusion

In this study, a two-machine shop floor scheduling problem multi-predecessor constraints is studied. In a manufacturing environment, the first machine may be a production stage for different kinds of materials, and the second machine may be the assembly stage using the materials from stage 1. This paper describes the *multi-predecessor constraint*, and proves that the complexity of F2|multi-predecessor|C_{max} is NP-hard in strong sense. A branch and bound algorithm is put forward to solve the problem. Some dominant rules are established to eliminate branches and improve efficiency.

Concerning jobs with *multi-predecessor constraints*, several issues are worthy of future research. The development of algorithms for the two-machine shop floor with other performance measures can be investigated. Polynomial algorithms for special types of problems can be developed. More dominant rules can be found for the branch and bound algorithm. Application of our results to more complex manufacturing environments, such as the multiple-stage shop floor problem, is another area of research.

References

[1] V. A. Strusevich (1997), Shop scheduling problems under precedence constraints, *Annals of Operations Research* **69**, 351 – 377.

[2] C. L. Monma (1979), The two-machine maximum flow-time problem with series-parallel precedence constraints: An algorithm and extension, *Operations Research* 27, 792-798

[3] Michael Pinedo (2002), *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Second Edition.

[4] A.M.A Hariri and C.N. Potts (1984), Algorithms for two-machine flow shop sequencing with precedence constraints, *European Journal of Operational Research* **17**(2), 238 – 248.

[5] Graham B. McMahon and Chong-John Lim(1993), The two-machine flow shop problem with arbitrary precedence relations, *European Journal of Operational Research* **64**(2), 249 – 257.

[6] Bongjin Gim, Guy L. Curry and Bryan L. Deuermeyer (1994), Two-machine flow-shop sequencing with sparse precedence constraints, *Computers & Industrial Engineering* **26**(1), 173 – 180.

[7] Chong John Lim and Graham B. McMahon(1994), The three-machine flow-shop problem with arbitrary precedence relations, *European Journal of Operational Research* **78**(2), 216 – 223.

[8] Alain Guinet and Marie Legrand(1998), Reduction of job-shop problems to flow-shop problems with precedence constraints, *European Journal of Operational Research* **109**(1), 96 – 110.

[9] Igor Averbakh, Oded Berman and Ilya Chernykh(2005), The m-machine flowshop problem with unit-time operations and intree precedence constraints, *Operations Research Letters* **33**(3), 263 – 266.

[10] Johann Hurink and Sigrid Knust (2001), List scheduling in a parallel machine environment with precedence constraints and setup times, *Operations Research Letters* **29**(5), 231 – 239.

Part III.

Abstracts

Statistical Quality Analysis of Schedulers under Soft-Real-Time Constraints

Hilbrandt Baarsma, Johann Hurink, Pierre Jansen
Universiteit Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands, h.e.baarsma@utwente.nl

This paper describes an algorithm to determine the performance of real-time systems with tasks using stochastic processing times. Such an algorithm can be used for guaranteeing Quality of Service of periodic tasks with soft real-time constraints. We use a discrete distribution model of processing times instead of worst case times like in hard real-time systems. Such a model gives a realistic view on the actual requirements of the system. The presented algorithm works for all deterministic scheduling systems, which makes it more general than most existing algorithms and allows us to compare performance between these systems. We show that the complexity of our algorithm is competitive with other algorithms that work for a wide range of schedulers.

1 Introduction

Many modern devices have to be able to process streams of data. These streams often consist of tasks arriving at regular intervals, where each task has to be processed within a fixed real time interval (RT). With the increasing use of RT systems, the techniques for building these systems have been described in many papers. In this paper we focus on single processor systems, but it is likely that the underlying ideas of these techniques can also be applied in a distributed RT environment. If several streams have to be processed on a single processor, there is a risk that some tasks may not complete before their deadlines. In hard real-time systems, this is not acceptable. However, with the increase of all kinds of multimedia devices, where occasionally missing a deadline would be tolerated, soft deadlines become more acceptable. In this context it is of importance to know how many deadlines are missed. Finding the number of missed deadlines is easy when dealing with deterministic processing times, but we run into problems when processing times are stochastic. Stochastic processing times are more realistic than using only worst case estimates. Using worst cases, may lead to an oversized system that, in reality, is idle most of the time. Therefore, if missing a deadline occasionally is acceptable, it becomes an important question how we can scale our system to achieve a performance level that we tolerate, in terms of missed deadlines.
Although fast algorithms exist to check if a given scheduling method will result in a deadline being missed, there are few efficient algorithms [1] [3] [4], to calculate the expected number of missed deadlines. This paper describes such an algorithm with three important features: it can take into account the effect of variation in processing times, it can compare hard vs. soft-real-time and it can compare a wide range of different scheduling techniques. The combination of these properties make it a valuable tool.

2 Analyzing performance using dynamic programming

We will use a dynamic programming algorithm to efficiently calculate the performance of a certain scheduling algorithm on a given set of tasks. We use the concept of states. Let $S_t = \{s_1^t, s_2^t, \ldots\}$ be the set of all possible states at time t. Each state $s \in S_t$ contains a list Q_s and the probability p_s of being in this state. The list Q_s consists of tasks that still need to be processed, paired with their remaining processing time. If we do not allow preemption, then the state needs two extra

variables, containing the task currently being scheduled and its remaining processing time.
To be able to evaluate how a scheduling method SM processes a task set Ω in a certain time period, we have to adjust every state, to reflect how the system evolves in time when scheduled by SM. For adjusting the states, the relevant events are when a new task enters the system or when a task reaches its deadline. In between two consecutive event times, t_i and t_{i+1}, no new states emerge. Only the values of the remaining processing times within the states change. Thus, the relevant times are given by the multiset $T = \{t_1, t_2, t_3, \ldots\}$ of all deadline and release times. This multiset has the property that $t_1 \leq t_2 \leq t_3 \leq \ldots$ and if times are equal, the deadline events are given before the release events. We define a function $F_{SM}(s_j^{t_i}, t_i, t_{i+1}) \rightarrow s_j^{t_{i+1}}$, that describes the change of the state $s_j^{t_i} \in S_{t_i}$ within the time interval $[t_i, t_{i+1}]$. The resulting state $F_{SM}(s_j^{t_i}, t_i, t_{i+1})$ belongs to $S_{t_{i+1}}$. Note, that the used function F depends on the scheduler used. If the set $S_{t_i} = \{s_1^t, s_2^t, \ldots s_k^t\}$ consists of all states at time t, we obtain $S_{t_{i+1}}$ by applying F_{SM} to all s_j's from S_{t_i}. The first thing F_{SM} does is to calculate $t_{i+1} - t_i$ and, if this is not zero, it updates the state according to SM. This means that some tasks in the new state have less load than they had in $s_j^{t_i}$. If the event at t_i is a deadline of a task τ_j, F_{SM} checks if τ_j has any load left. If this is the case, task τ_j has missed its deadline, thus, the task is removed from Q_s and p_s is added to the expected number of errors for τ_j. If F_{SM} has been applied to all states in S_{t_i}, we check if there are any identical states in the resulting set $S_{t_{i+1}}$. If this is the case, we merge them, i.e. if the states s_1 and s_2 have $Q_{s_1} = Q_{s_2}$, we set p_{s_1} to $p_{s_1} + p_{s_2}$ and then delete s_2 from S. Two different states can end up being identical for several reasons. For example, during $[t_i, t_{i+1}]$ both states end up being idle, which happens when those states have all their tasks ready before their deadlines. How they reached this point is no longer important now.
If the event at t_i is a release event of τ_j, then the output of F_{SM} is not one state, but several states. Each output state corresponds to a different realization of the processing time C_j of τ_j and the sum of the probabilities corresponding to these new states is equal to the probability of the original state.
By using F_{SM} on all states at every event, we eventually reach the starting state again. At this point, the hyperperiod, we know for each task how many deadlines we expect to miss over time.

3 Complexity and Testing

In this section we derive the worst case time complexity of the method we introduced and compare it with the complexity of the method in [1], since this method is the closest to our own algorithm. We also describe some of the results of tests with our algorithm.
Since feasibility analysis of an arbitrary periodic task system is shown to be co-NP-hard in the strong sense [2], it is not likely that there is a way to get an exact performance measure in polynomial time, even more so when we are dealing with stochastic processing times. Our algorithm has a total complexity of $O(qnm^n)$, where n is the number of tasks, m is the maximum processing time over all tasks, while q represents the total number of deadline and release events.
If we compare this to the $O(q^3m^2)$ complexity of [1], we see that if the number of events, q, is high, our performance can be much better. This is because although [1] claims a polynomial time performance, it fails to take into account q is exponential in n. In situations where m or n is high, our algorithm might start to perform worse. This difference in complexity of our algorithm and the algorithm of [1] is explained by the fact that the latter has a smaller state space. In their algorithm, the processing times of separate tasks can be added because the algorithm has to do a separate calculation for each deadline, while in our algorithm, the results are calculated in one run.
We implemented our algorithm in C++ to see how it deals in practice with varying values of n, m,

and other variables. We generated randomized task sets and tested how long our algorithm needs to complete a run. Our algorithm responds well to increases in m, showing an increase in runtime that is less than would be expected from the worst case complexity. Increasing n has a bigger impact on our algorithms performance and task sets consisting of more than ten tasks may cause our algorithm to run several days on a normal desktop computer. These long run times depend heavily on the kind of scheduling used though. We used EDF and RM scheduling, both with and without preemption. Not using preemption has a significant positive effect on the complexity of our algorithm, since it reduces the size of the state space.
We were able to test our algorithm's versatility as well. For example we developed a simple modification to RM and EDF to better deal with overload situations. We showed it is very effective in reducing the expected number of deadline misses in overload situations. We also showed that using stochastic processing times to model a task's behavior leads to a much better estimate of the processor demand when compared to hard real-time and worst case estimates. With twenty percent less processing power we still make more than 99 percent of the deadlines.

4 Conclusion

Using methods derived from dynamic programming, allows to check effectively how many missed deadlines we may expect in a periodic system with soft real-time constraints. Although run times are still exponential in the worst case, this algorithm generally offers a significant boost in speed. Our algorithm can be modified to work with both hard and soft deadlines or deal with almost any kind of scheduling method. We have assumed discrete distributions of the processing times with a finite number of realizations. We believe these assumptions allows us to closely approach reality. Future research needs to be done, to see what kind of distributions are good approximations for the run times of different tasks.

References

[1] Kanghee Kim and Jose Luis Diaz and Jose Maria Lopez (2005), An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations, *IEEE Trans. Comput.* **1**, 1460 – 1466.

[2] Joseph Y.-T. Leung and M. L. Merrill (1980), A note on preemptive scheduling of periodic, real-time tasks, *IEEE Trans. Comput.* **1**, 115 – 118.

[3] Mark K. Gardner (1990), *Probabilistic analysis and scheduling of critical soft real-time systems*, University of Illinois, Urbana Illinois.

[4] T.-S. Tia and Z. Deng and M. Shankar and M. Storch and J. Sun and L.-C. Wu and J. W.-S. Liu (1995), Probabilistic performance guarantee for real-time tasks with varying computation times, *RTAS '95: Proceedings of the Real-Time Technology and Applications Symposium*

Sequencing JIT Mixed Model Assembly Lines Under Station-Load and Part-Usage Constraints using Lagrangean Relaxations

Joaquín Bautista Valhondo
Nissan Chair ETSEIB, Universitat Politècnica de Catalunya, Avda. Diagonal 647, 7th floor, 08028 Barcelona, SPAIN, joaquin.bautista@upc.edu

Jordi Pereira Gude
Departament d'Organització d'Empreses, Universitat Politècnica de Catalunya, Avda. Diagonal 647, 7th floor, 08028 Barcelona, SPAIN, jorge.pereira@upc.edu

1. Introduction

Many production systems make use of assembly lines for mass production. An assembly line is made up of several workstations through which the work in progress on a product flows. The stations are linked together by a transport system, like a conveyor belt, whose mission is to supply materials to their main flow and to move the production items from one workstation to the next. In ideal circumstances, an assembly line is responsible of the production of a single product. In this context, manufacturing operations required by the product are divided into a set of tasks and each task is assigned to a single workstation. The set of tasks assigned to one workstation is known as workload, and the workers and robots assigned to each workstation have a constant time to develop the tasks. The highest workload defines the cycle time, and it determines the production rate of the line.

On the contrary, a mixed model assembly line assembles different units, but the similitude between each unit is high enough to consider each unit as identical during the assembly line design. Mixed model lines are very frequent in the automotive industry, where diversity of options inside a single product are a strong demand from the customers. Practitioners are then faced with two different problems. The first one, the balancing of the assembly line, is equivalent to the single model case, but mean task operation times are used to balance the line; the second one consists in sequencing the line in accordance to the differences between the units.

The latter problem has been tackled in the literature by proposing procedures to sequence the units in the line. In one hand, the initiatives from the Artificial Intelligence field have been interested in the feasibility of the final sequence according to workload constraints in one or several workstations. On the other hand, approaches from the Operations Research and Operations Management fields have been focused in the regularization of parts usage via the minimization of an objective function measuring the difference between the real sequence and an ideal one.

The present work is focused on the formulation and resolution of an approach combining both objectives, and extends a previous work found in [3]. The preliminary results from the new proposed procedure, based on the Lagrangean relaxation of the mathematical model of the problem are very promising, improving the previously reported results from [3] and solving to optimality 32 of the 36 instances found in the literature.

The rest of the work is structured as follows. In section 2 a brief literature review is presented. Section 3 is devoted to the proposed procedure to solve the problem, while section 4 exposes the results of the procedure comparing them with those previously reported. Finally, in section 5 the conclusions of the present work are put forward and several extensions are hinted.

2. State of the Art

The literature collects two main approaches to the sequencing of mixed model assembly lines, regarding the desired conditions of the sequence.

The first one can be traced back to the JIT production system in TOYOTA as documented by Monden, see [9]. The proposed objective is to find a sequence with option and consumption rates as regular as possible. Regularity is measured via the evaluation of the differences between mean consumption rates and the ideal rates. This proposal tries to blend the station loads of workstations, to establish a constant flow in the supply chain, and thus, to minimize stock on hand, space requirements on the assembly lines and other operative costs.

Several authors, see [10], presented the second one as is usually known as the Car Sequencing Problem (CSP). In this approach the objective is based on the fulfillment of several constraints required for proper operation. The approach is based on the assumption that different options required by the units affect station loads, and station loads must be taken into account. When multiple units requiring an option appear consecutively in a sequence, the workstations in charge of the required option may not be able to end their job in the allotted time, generating a situation that, in the best case, additional workforce can solve, but incurring in additional costs. To obtain sequences where this situation does not occur, the solution is forced to fulfill several constraints per option requiring that a given number of units with this option do not appear in a given number of consecutive positions of the sequence.

In [3], a mixed model is proposed and solved, combining previous works from both approaches:

a) The regularization of the sequence is based on production rates, see [6]. The proposal associates regularity with a discrepancy function between ideal due dates, related to ideal positions in the sequence, and real due dates, related to the real position in the sequence. Under the due dates given by the authors, the problem can be easily solved by the Earliest Due Date (EDD) rule. In the proposed procedure, the problem will be solved as an Assignment Problem as indicated in [7].

b) The car sequencing constraints as given in [2] are imposed as additional constraints to the model. The constraints are given in the form, maximum H_o units requiring option o, in every N_o consecutive positions in the sequence.

The original proposal to tackle the problem is to use a column generation approach to solve the mathematical model. The authors report results for a specially tailored dataset, due to the lack of previous references, indicating that the problem is capable of solving instances with up to 40 units and 3 different options to optimality, as well as providing lower and, sometimes, upper bounds to the optimal solution.

3. Solving the problem using Lagrangean Relaxations

Another possible approach to solve the mathematical model is to use a Lagrangean relaxation approach, [5]. The method is not only capable of giving lower bounds to the solution of the instances of the problem, but also to generate upper bounds during the bounding phase. After the bounding phase, the generated problem can also be used to generate new bounds to combine with an enumerative procedure like a Branch and Bound one.

Lagrangean relaxation approaches are based on relaxing certain constraints, usually the hard constraints of the problem, and insert them in the objective function with a penalizing factor. The new problem has an additional set of variables, known as Lagrange multipliers. When the Lagrange multipliers adopt a given value, the remaining problem can be solved efficiently. Obviously a new problem appears, associated to find the best possible set of Lagrange multipliers.

In the present work, the relaxed constraints are those originated by the CSP, and the relaxed problem is solved as an Assignment Problem (AP). The weight of the multipliers is obtained by a subgradient method [5], and the solution of each related AP instance is verified for feasibility. If the subgradient method is not sufficient to assess optimality, the procedure tries to solve the instance via Branch and Bound, using a modified Assignment Problem to calculate bounds for every partial solution.

4. Computational Experience

To assess the quality of the proposed procedure, the algorithm was programmed in C. The instances from [3] were used and the results provided by the presented procedure and their reported results are compared. Let us note that the instances have three special characteristics: (1) all instances have a solution, (2) all instances have maximum work load compared to the maximum possible work load given by the constraints and (3) the dataset consists in 36 instances with a number of units between 10 and 50, and a number of options between 3 and 7.

The Lagrangean relaxation procedure is able to obtain the optimal solution, and verify its optimality, in 19 of the 36 instances, in comparison to the 14 instances solved to optimality in [3]. When the Branch and Bound procedure is used, with a time limit of 15 minutes per instance, the procedure is capable of solving 32 instances with running times ranging from 0 to 700 seconds and a mean running time of 22 seconds using a 1,8 Ghz. Pentium IV computer.

5. Conclusions and Extensions

The proposed procedure seems to be comparable to the newest procedures found in the literature, [4]. As indicated in [3] it can also be used to generate cyclic sequences, very often used in practice. Further experiments with larger instances, with very different characteristics as those adapted from the CSPLIB (www.csplib.org), have failed to provide good solutions. To tackle bigger instances a different approach, as a metaheuristic, using the same ideas, is being developed.

Finally, it is important to cite that the current procedure can be used with different objective functions, as those given in [8]. Further work should be required to adapt the procedures to the proposals of [1] or [9], whose formulation resembles more clearly the desires of the automotive industry.

6. Aknowledgements

The authors acknowledge the Spanish Operation of Nissan as well as the UPC Nissan Chair for partially funding this research work. This research paper has also been partially funded by DPI2004-03475 Grant from the Spanish government.

References

[1] J. Bautista, R. Companys, A. Corominas (1996) Heuristics and exact algorithms for solving the Monden problem, *European Journal of Operational Research* **88**(1), 101 – 113.

[2] M. Dincbas, H. Simonis, P. van Hentenryck (1988), Solving the car-sequencing problem in constraint logic programming. *In Y. Kodratoff, editor, Proceedings ECAI-88*, 290 – 295.

[3] A. Drexl, A. Kimms (2001), Sequencing JIT Mixed-Model Assembly Lines Under Station-Load and Part-Usage Constraints, *Management Science* **47**(3), 480 – 491.

[4] A. Drexl, A. Kimms, L., Matthiessen (2006), Algorithms for the car sequencing and the level scheduling problem, *Journal of Scheduling* **9**, 153 – 176.

[5] M.L. Fischer (1981), The Lagrangean Relaxation Method for Solving Integer Programming Problems, *Management Science* **27**(1), 1 – 18.

[6] R.R. Inman, R.L. Bulfin (1991), Sequencing JIT mixed model assembly lines. *Management Science* **37**(7), 901 – 904.

[7] W. Kubiak, S. Sethi (1991), A note on Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science* **37**(1), 121 – 122.

[8] J. Miltenburg (1989), Level schedules for mixed-model assembly lines in just-in-time production systems, *Management Science* **32**(2), 192 – 207.

[9] Y. Monden (1983), *Toyota Production System* , Institute of Industrial Engineering Press.

[10] B.D. Parello, W. C. Kabat (1986), Job-Shop Scheduling Using Automated Reasoning: A Case Study of the Car-Sequencing Problem, *Journal of Automated Reasoning* **2**, 1 – 42.

Clustering Within Timetabling Conflict Graphs

Camille Beyrouthy, Edmund K. Burke, Dario Landa-Silva
School of Computer Science & IT, University of Nottingham, UK, {cbb, ekb, jds}@cs.nott.ac.uk
Barry McCollum, Paul McMullan
Department of Computer Science, Queens University of Belfast, UK, {b.mccollum, p.mccullan}@qub.ac.uk
Andrew J. Parkes[1]
School of Computer Science & IT, University of Nottingham, UK, ajp@cs.nott.ac.uk

A key concept in timetabling problems is that of the conflict graph with edges representing pairs of events that are not allowed to occur at the same time. Usually, the only information presented about such graphs is their density. However, intuitively, it seems likely that such graphs are structured, and likely to have some clustering. In analysing the structure of social networks or of the world-wide web it is common to use various measures. Amongst these is the "clustering coefficient". We propose using this coefficient to analyse timetabling conflict graphs, and give results showing that on some common benchmarks the graphs are indeed clustered by this measure.

1 Introduction and Context

In previous papers [1, 2, 3] we have studied the issue of space planning within academic institutions [5]. The problem is that currently teaching space is poorly utilised. In many institutions, teaching rooms are used only half the time, and even when used they are often only half full. Since building and maintaining rooms is expensive (the second highest institutional budgetary consideration after staff), it is not surprising that institutions would like to rectify this situation. On the other hand, excess teaching space is often requested in order to satisfy institutional timetabling requirements. This leads to the question of precisely how to manage the balance; making best use of minimal space whilst still satisfying demand. In addition, projected demand for teaching space must be considered when deciding upon future space requirements. We intend to address this by supporting institutional decision making with the following methodology. Firstly, analyse and quantitatively classify the current student enrollment and course structures. Secondly, in order to generate the test cases for analysis, create a simulator to generate course structures and student enrollments in a meaningful way. Finally, using the simulator together with appropriate course-timetabling software, run simulations under various proposed scenarios for space changes, and develop a methodology for evaluation and comparison of these scenarios. This outlined methodology is a form of "simulation optimisation" [6]. A crucial part of the process is having reasonable confidence that the simulator is realistic compared to real-world scenarios. The output of this research strives to ensure that the system is validated. The research will also explore the sensitivity of the final decisions to the assumptions underlying the simulator, to ensure that they are applicable.

In practice, this means that the simulator will be designed: (i) based on properties and patterns observable in existing instances, and (ii) validated against such patterns. However, there are few ways in order to compare simulated and real instances. Currently, the only property typically measured of a timetabling instance is the density, d, of the conflict graph. However the density is far too "blunt" a tool: graphs with similar densities might have very different structures. This suggests that better measures are needed in order to characterise and exploit the properties of conflict graphs. We aim to identify a suite of properties to measure in a timetabling instance. Such a suite will be used to ensure that the simulator uses instances that have realistic structures. Here,

[1]Contact Author. (Authors listed alphabetically)

Name	Exams (n)	edges (e)	Dens.(d) (%)	Clust. Coeff. c (%)
hec-s-92	81	2823	42	67.3
sta-f-83	139	611	14.4	85.8
yor-f-83	181	941	28.9	57.5
ute-s-92	184	2750	8.5	53.3
ear-f-83	190	1125	26.7	62.9
tre-s-92	261	4360	5.8	45.3
lse-f-91	381	2726	6.3	56.6
kfu-s-93	461	5349	5.6	56.6
rye-s-93	486	11483	7.5	60.2
car-f-92	543	18419	13.8	45.6
uta-s-92	622	21267	12.6	40.4
car-s-91	682	16925	12.8	40.8
pur-s-93	2419	30032	2.9	36.5

Table 1: Sizes and densities of the conflict graphs generated by the Carter instances, together with their clustering coefficients.

we start building such a suite by using standard concepts from the network analysis literature, specifically, the "clustering coefficient" of the conflict graph.

Many papers have studied the graphs resulting from social networks and the world-wide web, e.g. see [7]. One of the techniques used in such analysis is the "clustering coefficient" defined as follows. Suppose that the degree of node i is k_i, then there are potentially $k_i(k_i-1)/2$ edges between the neighbours of i. Let $c_i \in [0,1]$ be the local density of the graph between the neighbours of node i.

$$c_i = \frac{\text{number of edges between neighbours of } i}{k_i(k_i-1)/2} \tag{1}$$

The overall "clustering coefficient", $c \in [0,1]$, is defined as the mean value (with respect to the n nodes of the entire graph) of the clustering c_i

$$c = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{2}$$

In a random graph, the edges are selected randomly and independently with probability p [4], hence the expected density of the local neighbourhood is p, the same as the overall density. Here, we will say that the graph is clustered if the clustering coefficient, c, is higher than the overall density, d.

2 Empirical Clustering Coefficients

In the context of the conflict graph in timetabling problems, it is natural to expect that if an event A conflicts with events B and C, then the chances of B and C conflicting with each other is higher than the overall (average) density. This corresponds to expecting the conflict graph to be clustered. Table 1 gives the clustering coefficients of the standard *real-world* Carter instances of Exam Timetabling[2], and confirms that they do indeed have $c > d$, and so are clustered.

For the purposes of the "International Timetabling Competition (TTComp)"[3], *artificial* course timetabling problems were "designed by Ben Paechter for the Metaheuristics Network". Sixty

[2]See `ftp://ftp.mie.utoronto.ca/pub/carter/testprob/` and `http://www.cs.nott.ac.uk/~rxq/data.htm`
[3]`http://www.idsia.ch/Files/ttcomp2002/`

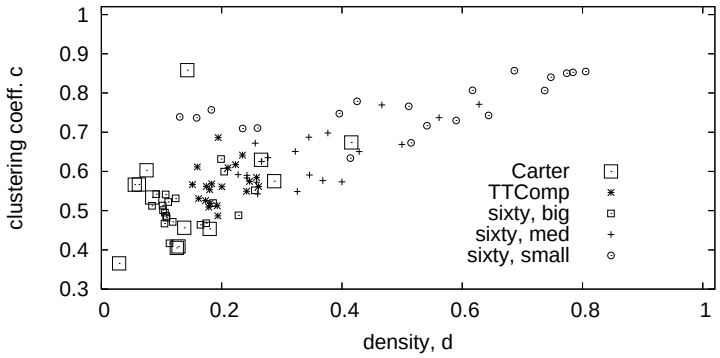


Figure 1: Each point corresponds to the conflict graph from a separate timetabling instance, plotted by their clustering coefficient, c, against their density, d.

more instances were also made publically available later and are grouped into the sets: "small", "medium" and "big". Figure 1 is a plot of clustering against density for the Carter and TTComp instances. It shows that the TTComp instances seem to fall into the same region as the Carter instances.[4] Note that the 'small' artificial instances seem to have unrealistic densities. However, it is reassuring to observe that the 'large' instances do fall into the same region of the (d, c) plane as the real Carter instances. We believe this supports the argument that the TTComp instances are a reasonable test for solvers. We observe that larger problems generally have both smaller density and clustering. However, whilst the density does not seem to have a lower limit, the clustering does not drop below about 40%. That is, even when there are few conflicts they still tend to cluster significantly, with the neighbourhood of nodes having a density of about 40%. Besides being a test of the validity of artificial instances, this might also have interesting implications for the design of solvers.

We have seen that timetabling conflict graphs are clustered and that the artificial instances used so far exhibit similar clustering. It is likely that the clustering will significantly affect important properties of the conflict graphs - for example, their chromatic numbers. Also, algorithms might be improved if they were to directly measure and exploit the clustering of the graphs. Current work is investigating such links and ways to exploit measured clustering. Finally, we remark that other domains such as scheduling and rostering generate ("conflict") graphs, and these might well also exhibit properties that could be revealed (and exploited) by their clustering coefficients. Our findings help us understand the structure of conflict graphs and so are an important step towards improving teaching space utilisation because timetabling directly impacts this issue.

References

[1] C. Beyrouthy, E.K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes (2006), The teaching space allocation problem with splitting, In *Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), to appear in LNCS*.

[4] One might object that the Carter instances are for *exam* timetabling and the TTComp for *course* timetabling. However, we expect that the conflict matrices will have related structures, as students taking exams presumably also took the associated course. However, this issue will be under future study.

[2] C. Beyrouthy, E.K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes (2006), Towards improving the utilisation of university teaching space, Technical Report NOTTCS-TR-2006-5, School of Computer Science & IT, University of Nottingham.

[3] C. Beyrouthy, E. K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes (2006), Understanding the role of UFOs within space exploitation, In *Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006).*

[4] B. Bollobas (1985), *Random Graphs*, Academic Press, London, England, 1985.

[5] B. McCollum and P. McMullan (2004), The cornerstone of effective management and planning of space, Technical report, Realtime Solutions Ltd, Jan 2004.

[6] J.R. Swisher, P.D. Hyden, S.H. Jacobson, and L.W. Schruben (2000), A survey of simulation optimization techniques and procedures, In J. Joines, R. Barton, K. Kang, and P. Fishwick, editors, *Proceedings of the Winter Simulation Conference.*

[7] D.J. Watts and S.H. Strogatz (1998), Collective dynamics of 'small-world' networks, *Nature* **393**, 440 – 442.

Dynamic Cooperative Search for Constraint Satisfaction and Combinatorial Optimization: Application to a Rostering Problem

Boris Bontoux, Dominique Feillet
Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse, 339 chemin des Meinajaries, Agroparc B.P. 1228, F-84911 Avignon Cedex 9, France, {boris.bontoux, dominique.feillet}@univ-avignon.fr

Christian Artigues
LAAS CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France, artigues@laas.fr

Eric Bourreau
LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France, eric.bourreau@lirmm.fr

In this paper, we are interested in enumerative resolution methods for combinatorial optimization (COP) and constraint satisfaction problems (CSP). We introduce a new approch for the management of branching, called Dynamic Cooperative Search (DCS) inspired from the impact-based search method proposed in [7] for CSPs. This method defines in a dynamic way priority rules for variable and value selection in the branching scheme. These rules are meant to be independent of the considered problem. As in [7], the principle is to take into account, through learning methods, of the impact of branching decisions in already explored subparts of the search tree. We show the interest of DCS on a real-life rostering problem.

1 Dynamic Cooperative Search: principle and first results

Most of the exact resolution methods for combinatorial optimization (COP) and constraint satisfaction problems (CSP), such as tree search, branch and bound and dynamic programming, rely on an intelligent enumeration of the solutions. In tree search methods, used for the resolution of both CSPs and COPs, child generation (or branching) consists in selecting a variable and its tentative value. In many cases, the branching policy, i.e., the strategy for the variable and value selections, have a huge influence on the efficiency of the method. In the case where the strategy allows one to obtain quickly good solutions, the enumerative method can be used within a limited amount of computing time, which is often imposed in practical applications such as the industrial rostering problem considered in this paper.

The Dynamic Cooperative Search (DCS) method is designed for leading the search towards the subparts of the solution space which contain the (best) solutions. The underlying ideas for speeding up the search tree exploration are not recent. Several techniques have been proposed in this way, most of them relying on a exploration of the search tree limited by a heuristic mechanism, such as Beam Search [6], Limited Discrepancy Search [2], Minimal Discrepancy Search [4], Branch-and-Greed [8]. Some of the efficient implementations of these methods have the common characteristic that they base the heuristic criteria on the structure of the studied problem. The principle of the DCS method is, on the opposite, relatively independent of the studied problem.

Refalo has proposed in [7] a general purpose search strategy for constraint programming based on *impacts*. He defines the impact of an assignement of a variable x_i to a value a as $I(x_i = a) = 1 - P_{after}/P_{before}$ where P is an estimation of the size of the search tree. To reduce the search tree, a variable with the greatest impact should be choosen first while a value with the smallest impact should be used first. In [5], an extension of this method to weighted CSPs is proposed.

The DCS method proposes a generalization of this method for both CSPs and COPs. It is used in a depth first search scheme and defines for every node priority rules for variable and value ordering. These orders are deduced from the characteristics of subtrees obtained from the previous

branching on the considered variables. Each variable is associated with a dynamic weight and the dynamic ordering of its values (represented through value weights). During the exploration, at each sub-tree closure (when the branch from the last value of the variable has been explored), the variable weight and the ordering of its values are updated. The way the variable and value weights are updated is a parameter of our method and depends on whether we are solving a CSP or a COP. In what follows we give, for each case, a possible implementation.

For COPs, we can define the weight of both variables and values as the cost of the best solution found in the considered subtree. Other possibilities is to consider the number of node prunings (for a Branch and Bound), the mean value of the solutions... Following this principle, we first apply DCS to the well-known Traveling Salesman Problem for which a huge number of solutions can be found quickly. We use a naive branching scheme where each node is associated with a city i and each child node corresponds to the city j visited immediately after i. Here we associate a weight w_{ij} to each arc (i, j) as the value of the best solution found in all its explored subtrees. The branching scheme consists of selecting the candidate arcs in the decreasing weight order. Under the above-described scheme, this correspond to a fixed variable ordering (each node of level k corresponds to fixing a value to variable x_k stating which arc is taken in position k) while values are explored through the DCS principle. Arc weights are initialized through a preprocessing technique described below.Preliminary results show a significant reduction of the size of the search tree and of the consumed CPU time against a simple Depth First Search method.

The Table 1 shows our results on a benchmark of 60 TSP instances with a number of cities varying from 10 to 22. The results presented are the mean CPU time (in seconds) and number of nodes, over all instances.

	Depth First Search	DCS	DCS with random solutions
Number of nodes	6,089,428	4,790,007	3,258,450
CPU time (in sec)	57	45	30

Table 1: Computational results for several methods applied to the TSP

For CSPs, the method is based on the following rules. When a variable is set to a value, we remember as the weight of this value, the maximum depth of the obtained subtree (after applying constraint propagation). At the branching step, once a variable is selected, its values are considered in the decreasing weight order aiming at first exploring the most promising regions. The weight of a variable is equal to the mean weight of its values. At the branching step, the variable are considered in nondecreasing order of their weights, aiming at reducing the size of the sub-tree, as in the impact-based search proposed in [7].

For both COP and CSP, a preprocessing phase is applied, during which severals random solutions are built to initialize the weights. For CSP, a tree search is applied with a random selection of variables and values. Weights are then computed with the method described above. For COP, a large number of solutions is randomly generated in a tree search and the variable and value weights are set with the obtained solution costs. The goal of a preprocessing phase is to lead the search towards the sub parts of the solution space which may contains a (good) solution, as a basic learning process. The time allowed for the preprocessing phase is limited to 10% of the computing time.

2 Application to an industrial rostering problem

The considered industrial problem is a personnel scheduling problem with constraints, similar to the *nurse rostering problem* [1]. Employees may have several skills. The problem is a constraint satisfaction problem aiming at satisfying skill constraints while ensuring that employees who have several skills are employed on each skill in a balanced way. This problem is a real case, proposed in the context of a partnership with the Daumas Autheman et Associés IT company (Aix-En-Provence, France) and concerns employee rostering in a ferry unloading company.

Let n be the number of employees. X_{ijk} denote the activity of employee i, on day j and week k, with $i = 1, \ldots, n$, $j = 1, \ldots, 7$, $k = 1, \ldots, l$. X_{ijk} is equal to 0 when the employe is off duty. $S = 1, \ldots, t$ denotes the set of considered activities. Let $C_i \subseteq S$ for $i = 1, \ldots, n$ be the set of activities employee i is able to perform. E_{mjk} gives the number of required employees for activity m on day j and week k. The constraints are the following :

$$|\{X_{ijk}|X_{ijk} = m\}| = E_{mjk} \quad j = 1, \ldots, 7 \quad k = 1, \ldots, l \quad m = 1, \ldots, t \tag{1}$$

$$|\{X_{ijk}|X_{ijk} = 0\}| \geq 2 \quad i = 1, \ldots, n \quad k = 1, \ldots, l \tag{2}$$

$$|\{X_{i6k}|X_{i6k} = 0\}| \geq r \quad i = 1, \ldots, n \quad k = 1, \ldots, l \tag{3}$$

$$|\{X_{i7k}|X_{i7k} = 0\}| \geq r \quad i = 1, \ldots, n \quad k = 1, \ldots, l \tag{4}$$

$$\lfloor \frac{5}{|C_i|} \rfloor \leq |\{X_{ijk}|X_{ijk} = m\}| \leq \lceil \frac{5}{|C_i|} \rceil \quad m \in C_i \quad i = 1, \ldots, n \quad k = 1, \ldots, l \tag{5}$$

$$X_{ijk} \in C_i \quad i = 1, \ldots, n \quad j = 1, \ldots, 7 \quad k = 1, \ldots, l \tag{6}$$

Constraints (1) state that, on each time period, the number of employees who are assigned to activity m is equal to the required number of employees. Constraints (2) ensure that each employee is off duty at least two days a week. Constraints (3) and (4) ensure that the number of vacant Saturdays and Sundays is greater than r (where r is a data). Contraint (5) state that if an employee has several skills, he must be employed for each activity in a balanced way, 5 being the average working days per week.

We have implemented the variant of DCS as stated in Section 1 for CSPs and compared it with standard methods (goals) of the ILOG Solver constraint programming package.

The Table 2 shows our results on a benchmark of 25 randomly generated instances with a number of weeks varying from 4 to 75. The results presented are the mean CPU time (in seconds) to obtain a solution. The time limit is fixed to 600 seconds. Our method has been compared with two default goals in Ilog Solver (First Unbound, which chooses the first unbound variable first and Min Domain, which choose the unbound variable with the smallest domain first). The "Fisrst Unbound" and "Min Domain" rows display the results of these strategies with the smallest value selection rule (column ILOG SOlver) and the DCS value selection rule (column "DCS"). The "Minimum average Weight" row displays the result for the full DCS variable and value strategies.

	Ilog Solver	DCS
First Unbound	94 s	117 s
Min Domain	52 s	24,34 s
Minimum Average Weight		0,75 s

Table 2: Computational results for several methods applied to the Rostering Problem

These results show that our method permits to solve much more instances than the default goal in Ilog Solver and it is up to 50 times faster for instances solved by both methods.

References

[1] B. Cheang, H. Li, A. Lim, and B. Rodrigues (2003), Nurse rostering problems – A bibliographic survey, *European Journal of Operational Research* **151**, 447 – 460.

[2] W.D. Harvey and M.L. Ginsberg (1995), Limited discrepancy search *In Proceedings of IJCAI95*, 607 – 613.

[3] J. Hooker (2000), Logic-Based Methods for Optimization - Combining Optimization and Constraint Satisfaction. Wiley-Interscience series in discrete mathematics and optimization. John Wiley and Sons.

[4] W. Karoui , M.J. Huguet , P. Lopez , W.Naanaa (2006), MDS: a learning method based on discrepancies and propagations, *Rapport LAAS N06093*.

[5] N. Levasseur, P. Boizumault, S. Loudni (2007), Une heuristique de sélection de valeur dirigée par l'impact pour les WCSP, *Journée 19/01/2007 GT Contraintes et RO*.

[6] P. S. Ow, T. E. Morton (1988), Filtered beam search in scheduling, *International Journal of Production Research* **26**, 35 – 62.

[7] P. Refalo (2004), Impact-Based Search Strategies for Constraint Programming, *CP 2004*, 557 – 571.

[8] F. Sourd, P. Chretienne (1999), Fiber-to-Object Assignment Heuristics, *European Journal of Operations Research* **117**.

Selecting and Scheduling Tasks with Agreeable Time Windows and Setup Costs

Philippe Chrétienne, Francis Sourd
Universit Paris 6, LIP6, France, {Philippe.Chretienne, Francis.Sourd}@lip6.fr

1 Introduction

The problem studied in this paper originates in a satellite launcher problem. When a satellite launch is ordered to a launching company, three important parametrers have to be taken into account : the time window during the satellite must be launched, the satellite mass and the cost of the satellite launch. When a given set of launches has to be planned, the company has to decide which satellites will be launched and when these selected launches will be scheduled, with the objective of maximizing its profit. When a non empty set of satellite launches is assigned to a time slot, the company has to pay for the installation of a launching platform. Moreover, the total mass of the launched satellites must not exceed the launching capacity of the platform.

In a formal description of the problem, we consider n unit-length independent jobs $J_1, \cdots, J_n$ each of which represents a satellite and has an individual weight w_i (to be understood as the weight of an item in a knapsack problem). For each job J_i, it must be decided whether it will be processed or not and, in the positive case, in which time-slot of an a priori given time window $[r_i, d_i]$ it must be performed. A positive *gain* g_i results from deciding to perform J_i and a positive fixed *setup* cost K_t is due if time slot $[t-1, t]$ (also denoted by t) is assigned to at least one job. Finally, the sum of the weights of the jobs performed in any time-slot must not exceed a given capacity C. A solution S of the problem is a pair (E, s) where :

- E is the subset of the executed jobs,
- $s : E \mapsto T$ indicates the time-slots assigned to the jobs of E (to simplify the notation, we shall write $s(i)$ for $s(J_i)$,
- for any $J_i \in E$, $s(i)$ must be in $[r_i + 1, d_i]$
- for any time-slot t in $s(E)$, the sum $\sum_{\{j|s(j)=t\}} w_j$ must be less than C.

The cost of the solution (E, s) is $\sum_{t \in s(E)} K_t - \sum_{j \in E} g_j$ (the opposite of this expression is of course the total benefit) and the problem is to find a minimum-cost solution.

Our problem is clearly NP-complete since finding if there is a schedule with makespan less than 2 such that all the jobs are processed can be reduced to the PARTITION problem. We can also reduce 3-PARTITION to our problem, which proves that it is strongly NP-hard. The goal of this work is to study some polynomial special cases. Some cases are special cases of classical scheduling problems.

- The single-machine case, that is $C = 1$ and $w_j = 1$ for all j is easy. As at most one job can be scheduled in any time slot, we simply have an assignment problem between jobs and time-slots where the cost to assign slot t to job J_j is given by $K_t - g_j$.
- When there are no setup costs, that is $K_t = 0$ for all t, the objective function is equivalent to the sum of tardy jobs since a tardy job can be regarded as a non-processed job. Van den

Akker and Hoogeveen propose a survey of these problems [2]. Other authors have also studied the presence of rejection penalties in addition to other scheduling costs such as the weighted flow time [3] or the sum of tardiness [1].

In this paper we propose a dynamic-programming polynomial algorithm for the special case with unit weights, a common gain (i.e: $g_j = g$) and agreeable time windows. We recall that time windows are agreeable if $r_i < r_j \Rightarrow d_i \leq d_j$. It means that jobs are numbered in the order of the release dates (and deadlines).

2 The dynamic programming algorithm

We first propose a simple dominance property that is valid even without the assumption of agreeable time windows.

Property 1 *There is an optimal solution (E, s) such that for any two jobs J_i and J_j in E such that $s(i) < s(j)$, then either $d_i \leq d_j$ or $s(i) + 1 \leq r_j$.*

Proof. — Assume that an optimal solution (E, s) is such that for two jobs J_i and J_j in E we have $s(i) < s(j)$, $d_i > d_j$ and $s(i) + 1 > r_j$. By exchanging the time slots of the jobs J_i and J_j, we still get an optimal solution. We may then repeat the above transformation until the property is satisfied. ∎

The next dominance property basically relies on the dominance property:

Property 2 *There is an optimal solution (E, s) such that $i < j \Rightarrow s(i) \leq s(j)$.*

Proof. — Assume that an optimal solution (E, s) is such that for two jobs J_i and J_j in E we have $s(i) > s(j)$ and $i < j$. From the agreeable time windows assumption, we have $r_i \leq r_j < s(j) < s(i) \leq d_i \leq d_j$. By exchanging the time slots of the jobs J_i and J_j, we still get an optimal solution. We may then repeat the above transformation until the property is satisfied. ∎

Let $(a_0, \cdots, a_q)$ be the ordered list of the distinct release times and deadlines. Clearly, $q < 2n$.

Property 3 *There is an optimal solution (E, s) such that if the jobs J_i and J_j are scheduled in the time interval $[a_{t-1}, a_t]$, then any job J_k with $i < k < j$ is also scheduled in this interval.*

Proof. — Consider an optimal schedule satifying Property 2. Assume that J_k is not processed in $[a_{t-1}, a_t]$ while J_i and J_j $(i < k < j))$ are scheduled in this interval. From Property 2, we know that J_k is not processed. Since there is no deadline in the interval, J_k is available at any time point. Therefore we can replace J_j by J_k and eventually reorder the jobs according to their index. We may then repeat the above transformation until the property is satisfied. ∎

As a consequence of this property, the set of jobs processed in any $[a_{t-1}, a_t]$ is an interval of jobs $J_i, J_{i+1}, \cdots, J_j$. These $j - i + 1$ jobs are clearly scheduled in $n_{ij} = \lceil (j - i + 1)/C \rceil$ time-slots which correspond to the smallest n_{ij} values of the setup costs in this interval. Therefore, the associated cost only depends on the number of jobs in the interval and is accordingly denoted by $\kappa_t(j - i + 1)$. By convention, if $j - i + 1 > (a_t - a_{t-1})C$, then we cannot schedule all these jobs and we set $\kappa_t(j - i + 1) = +\infty$.

For any $t \in \{0, \cdots, q\}$ and any $j \in \{0, \cdots, n\}$, we define the subproblem $\Pi(j, t)$ as follows: the jobs of $\Pi(j, t)$ are $\{J_1, \cdots, J_j\}$ and for each job J_i, its release time is still r_i and its deadline becomes $\min\{a_t, d_i\}$. Note that $\Pi(n, q)$ is the original problem.

Along with subproblem $\Pi(j,t)$, we define $\pi(j,t)$ to be the value of an optimal solution of $\Pi(j,t)$ and we show that $\pi(j,t)$ satisfies a recurrence equation.

Let us first observe that an optimal solution of $\Pi(j,t)$ satisfies Property 3. Therefore, in such an optimal solution, the jobs scheduled in $[a_{t-1}, a_t]$ are $J_i, \cdots, J_{i'}$. If $d_j < a_t$, then $d_j \leq a_{t-1}$ and no job among $\{J_1, \cdots, J_j\}$ can be scheduled in the last interval $[a_{t-1}, a_t]$. Therefore $\pi(j,t) = \pi(j,t-1)$.

Let us now assume that $d_j \geq a_t$. If $i' \neq j$, we can get a new optimal schedule by replacing the last $i' - i + 1$ jobs by the jobs $J_{i-i'+j}, \cdots, J_j$. Therefore, if there are l jobs in the last interval $[a_{t-1}, a_t]$, then the cost of the schedule is given by $\pi(j-l, t-1) + \kappa_t(l)$. Since all the jobs in $[a_{t-1}, a_t]$ must be release before a_{t-1}, we have:

$$\pi(j,t) = \begin{cases} \pi(j,t-1) & \text{if } d_j < a_t \\ \min\{\pi(j-l,t-1) + \kappa_t(l) \,|\, 0 \leq l \leq j \text{ and } r_{j-l+1} \geq a_{t-1}\} & \text{if } d_j \geq a_t \end{cases}$$

We clearly have $\pi(j,0) = 0$ for all j and $\pi(0,t) = 0$ for all t. From these values, the dynamic program can be run.

All the functions κ_t are piecewise constant and can be built in $O(T \log T)$ time where $T = a_q$ is the horizon of the schedule. Each value $\pi(j,t)$ can be computed in $O(n)$ time, which means that all the values are computed in $O(n^3)$ time. Therefore, the time complexity of the algorithm is $O(T \log T + n^3)$. In the special case where all the setup costs are equal to a common constant K, the values $\kappa_t(l)$ can be evaluated in constant time and the time complexity is in $O(n^3)$.

References

[1] Philippe Baptiste and Claude Le Pape, Scheduling a single machine to minimize a regular objective function under setup constraints, *Discrete Optimization* **2**, 83 – 99.

[2] M. van den Akker and H. Hoogeveen (2004), Minimizing the number of tardy jobs, in J.Y-T. Leung (ed), *Handbook of Scheduling*, chapter 12.

[3] D.W. Engels, D.R. Karger, S.G. Kolliopoulos, S. Sengupta, R.N. Uma and J. Wein (2003), Techniques for scheduling with rejection, *Journal of Algorithms* **49**, 175 – 191.

An Educational Tool for the Resource-Constrained Project Scheduling Problem

Filip Deblaere, Erik Demeulemeester, Willy S. Herroelen
Research Center for Operations Management, K.U. Leuven, Naamsestraat 69, 3000 Leuven, Belgium,
{filip.deblaere, erik.demeulemeester, willy.herroelen}@econ.kuleuven.be

Abstract
The well-known resource-constrained project scheduling problem (RCPSP) involves the determination of a baseline 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. We have developed an educational tool that visualizes (amongst others) project networks, baseline schedules, resource profiles and Gantt charts. The tool also features a number of known scheduling algorithms for solving the RCPSP, including an exact branch-and-bound procedure. Project networks can either be read in from files or be built up from scratch, using an intuitive graphical user interface.

1 The basic RCPSP

The basic RCPSP [4] involves a project network $G(N, A)$ with a set N of $n+1$ nodes representing the project activities. We assume a dummy start activity 0 and a dummy end activity n, which denote the start and the finish of the project, respectively. Both dummies have a duration equal to zero and do not consume any resources. The activities in the network are subject to zero-lag finish-start precedence constraints $(i, j) \in A$, indicated by the arcs of the network. We assume the presence of m renewable resource types, with a per period availability a_k, $k \in K$ with $K = \{1, \ldots, m\}$. The project activities i require an integer per period amount r_{ik} of resource type k, $k \in K$. A solution to the RCPSP consists of a vector of start times s_i, $i \in N$ such that the resource and precedence constraints are satisfied, and the project makespan s_n is minimized.

Feasible schedules for the RCPSP can easily be obtained by a so-called schedule generation scheme [7, 1], using an activity list – usually ordered according to a certain priority rule – as input. Numerous procedures have been developed for solving the RCPSP, both heuristic and optimal. The most successful exact procedure for the RCPSP appears to be dedicated branch-and-bound [2, 3].

2 An educational tool

An educational tool for visualizing the RCPSP has been developed in C++. It has a window-based Graphical User Interface (GUI) and runs on all Win32® platforms. The software is capable of handling projects consisting of any number of activities and any number of resource types. Project files in the "rcp" format (this is the format used in the well-known project scheduling library PSPLIB [8]) can be read in. Alternatively, the user can build up a project network from scratch, starting with an empty project and adding activities and precedence relations one at the time. These networks can then be exported to a file in the aforementioned "rcp" format. Early and late start schedules can be calculated, and the slack values of different activities can be graphically displayed under the form of a Gantt chart. The software also supports the calculation of various project statistics, such as resource strength, order strength and coefficient of network complexity, to name a few. A screenshot of the software is presented in Figure 1.

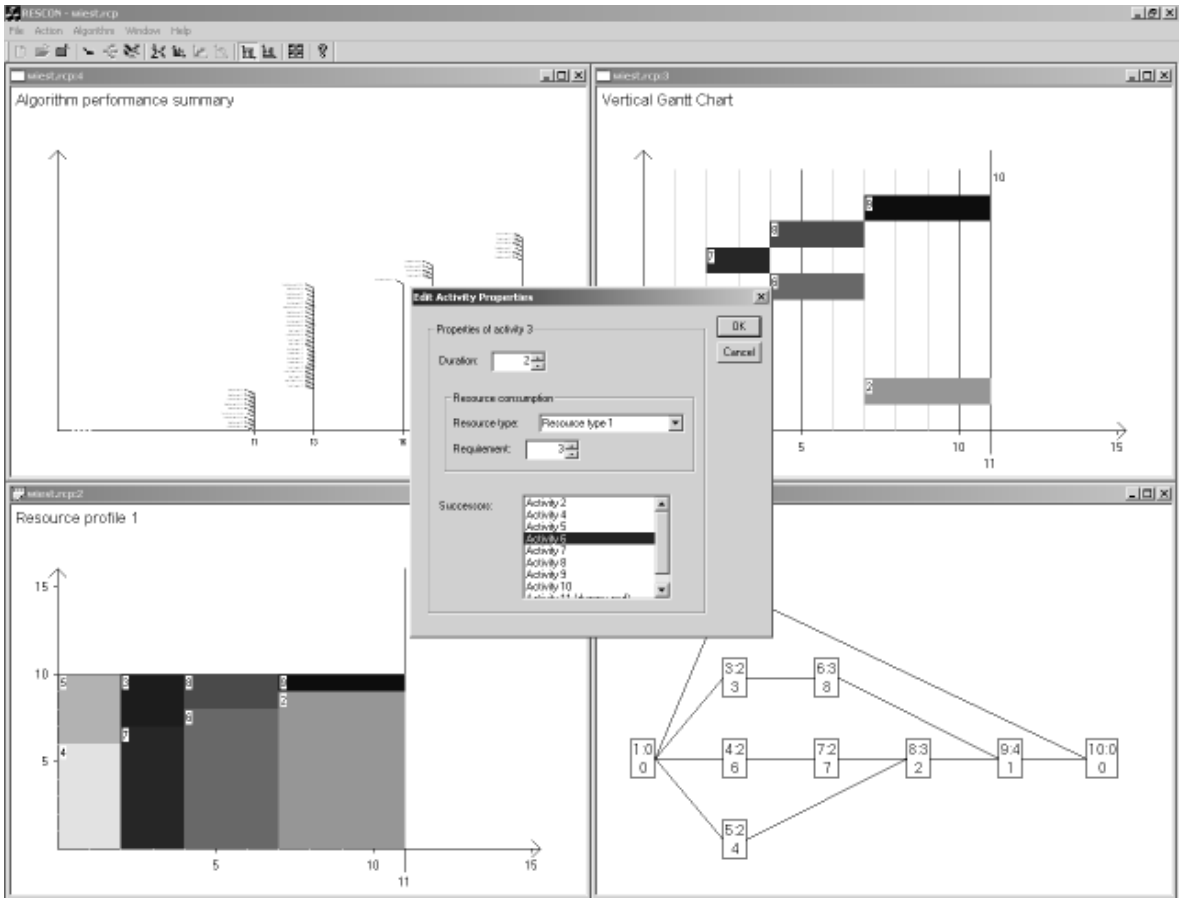


Figure 1: Screenshot of the software

Feasible schedules can be calculated using a large number of constructive heuristics. The software supports both the serial and the parallel schedule generation scheme, in combination with forward, backward or bidirectional planning. The user can choose from eight popular priority lists (e.g. latest finish time, minimum slack, ...), yielding a total of 48 heuristics to calculate a feasible schedule. We have also implemented an exact procedure for solving the RCPSP. It is a branch-and-bound procedure based on the procedure by Demeulemeester and Herroelen (1992), yielding optimal solutions in a relatively short computation time (provided the number of activities is limited). The software provides the possibility of summarizing the results (i.e., the obtained makespan) of the 48 heuristics plus the optimal procedure graphically, on a single timeline.

Also, for educational purposes, we provide the opportunity for "what-if" analysis: project parameters (such as activity durations, resource availability, precedence constraints) can be changed and the schedule can be recalculated, such that the immediate effects on the baseline schedule are just one click away.

As the software is still under development, no version of the software has been made publicly available yet. Because now, the user only has the opportunity to choose between rather poor heuristics on the one hand (the list scheduling procedures), and an optimal but computationally demanding branch-and-bound procedure on the other hand, we plan to incorporate a Tabu Search procedure [5, 6] in the software to solve the RCPSP in a heuristic way, in an attempt to strike a

balance between computation time and solution quality.

References

[1] Brooks, G. and White, C., (1965), An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem, *Journal of Industrial Engineering*, January-February Issue, 34 – 40.

[2] Demeulemeester, E. and Herroelen, W. (1992), A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem, *Management Science* **38**, 1803 – 1818.

[3] Demeulemeester, E. and Herroelen, W. (1997), New Benchmark Results for the Resource-Constrained Project Scheduling Problem, *Management Science* **43**, 1485 – 1492.

[4] Demeulemeester, E. and Herroelen, W. (2002), *Project scheduling - A research handbook*, 49 of International Series in Operations Research & Management Science. Kluwer Academic Publishers, Boston, MA.

[5] Glover, F. (1989), Tabu search, Part I, *INFORMS, Journal of Computing* **1**, 190 – 206.

[6] Glover, F. (1989), Tabu search, Part II, *INFORMS, Journal of Computing* **2**, 4 – 32.

[7] Kelley, J. (1963), The Critical-Path Method: Resources Planning and Scheduling, in Muth, J. and Thompson G. (Eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, 347 – 365.

[8] Kolisch, R. and Sprecher, A. (1997), PSPLIB - A Project Scheduling Library. *European Journal of Operational Research* **96**(1), 205 – 216.

Scheduling Resumable Deteriorating Jobs

Stanisław Gawiejnowicz
Adam Mickiewicz University, Umultowska 87, 61-614 Poznań, Poland
stgawiej@amu.edu.pl

Alexander Kononov
Sobolev Institute of Mathematics, Prospekt Akademika Koptyuga no. 4, 630090 Novosibirsk, Russia
alvenko@math.nsc.ru

1 Introduction

We consider the following single-machine time-dependent scheduling problem with constraints on the machine availability. There is given a set of independent and deteriorating jobs $J_j, 1 \leq j \leq n$, to be processed on a single machine, starting from time $t_0 > 0$. The processing time p_j of job J_j is a proportional function of the starting time S_j of this job, $p_j = \alpha_j S_j$, where $S_j \geq t_0$ and deterioration rate $\alpha_j > 0$ for $1 \leq j \leq n$. The machine is not continuously available and there are h disjoint periods of the machine non-availability. These periods are described by time intervals $[W_{i,1}, W_{i,2}]$, where $t_0 < W_{1,1}$ and $W_{i,1} < W_{i,2}$ for $1 \leq i \leq h$. Since in real-world applications the number of non-availability periods is usually small, we will assume that $h < n$. Moreover, since any such non-availability period can interrupt the processing of any job, we will assume that the jobs are *resumable*, i.e. if a job has been interrupted by the start time of a non-availability period, then this job does not need to be restarted and can be completed after the machine becomes available again. The applied criterion of schedule optimality is the maximum completion time $C_{\max} = \max\limits_{1 \leq j \leq n} \{C_{[j]}\}$, where $C_{[j]}$ denotes the completion time of the j-th job in schedule. For simplicity of presentation, the above problem will be denoted in short by $RDP(h)$.

The problem $RDP(h)$, introduced by Wu and Lee [7] for $h = 1$, is a combination of time-dependent scheduling with scheduling on a machine with non-availability periods. Time-dependent scheduling has numerous practical applications, e.g. in financial management and modelling service operations. Scheduling on machines with non-availability periods is applied in manufacturing environments in which machines have conservation or maintenance breaks, caused by production or service reasons. We refer the reader to reviews by Cheng et al. [1] and Lee [5] for more details on time-dependent scheduling and scheduling on machines with non-availability periods, respectively.

In this talk, first we extend some of the results obtained by Gawiejnowicz [2] and Ji et al. [4], where a counterpart of the problem $RDP(1)$ with *nonresumable* jobs has been considered. In particular, we prove the ordinary $\mathcal{NP}$-hardness of the $RDP(1)$ problem and thus we give an answer to the open problem stated in [4]. Next, we present a pseudopolynomial dynamic programming algorithm, DP, for the problem $RDP(h)$ in the case when h is a fixed number. After that we show that the algorithm DP allows us to construct an FPTAS for the $RDP(1)$ problem. Finally, we prove that for the problem $RDP(h)$ with $h \geq 2$ there does not exist a polynomial-time approximation algorithm with a constant worst-case ratio, unless $\mathcal{P} = \mathcal{NP}$.

2 Results

Let a machine, starting from time t_0, execute n jobs without idle times, let the processing times of the jobs be in the form of $p_j = \alpha_j S_j$ for $1 \leq j \leq n$, and let $[W_{h,1}, W_{h,2}], h \geq 1$, be the last interval

such that the machine executes some jobs after $W_{h,2}$. Let $W_{0,1} = 0$, $W_{0,2} = t_0$ and $k_0 = 0$. Then

$$C_{[n]} = \sum_{i=0}^{h}(W_{i,2} - W_{i,1}) \prod_{j=k_i+1}^{n}(1 + \alpha_{[j]}). \quad (1)$$

where k_i is the index of the job which was interrupted by the start time of the non-availability period $[W_{i,1}, W_{i,2}]$, $0 \leq i \leq h$.

Throughout the talk, the job which has been started before and completed not earlier than the start time of a non-availability period will be called a *critical job*, i.e. if for some $1 \leq j \leq n$ and $1 \leq i \leq h$ there hold inequalities $S_{[j]} < W_{i,1}$ and $C_{[j]} \geq W_{i,1}$, then $J_{[j]}$ is a critical job.

First we formulate the following two properties of an optimal schedule of the problem $RDP(h)$.

Property 1 *For the problem $RDP(h)$ there exists an optimal schedule such that jobs start in nondecreasing order of their processing times inside time interval $[W_{i-1,2}, W_{i,1})$ for $1 \leq i \leq h$.*

Property 2 *For the $RDP(1)$ problem there exists an optimal schedule such that the job $J_{\max}$ with deterioration rate $\alpha_{\max} = \max_{1 \leq j \leq n}\{\alpha_j\}$ is the critical job.*

Now we pass to the presentation of our results. We start with the proof of $\mathcal{NP}$-hardness of the problem $RDP(1)$. This proof is based on Property 2. The main idea is as follows. Let $t_0 = 1$ and let $J_{\max}$ be the critical job. We can separate the set of all jobs except $J_{\max}$ into two sets, N_1 and N_2. Set N_1 contains the jobs which are executed in an optimal schedule before the critical job and set N_2 contains the remaining jobs. The completion time of all jobs from N_1 does not depend on permutation of these jobs and is equal to $C' = \prod_{J_j \in N_1}(1+\alpha_{[j]})$. Moreover, C' has to be smaller than W_{11}. We can rewrite (1) as $C_{[n]} = (1 + \alpha_{\max}) \prod_{J_j \in N_1}(1 + \alpha_j) + (W_{i,2} - W_{i,1}) \prod_{J_j \in N_2}(1 + \alpha_j)$. Since the first term is a constant for any given instance of the problem $RDP(1)$, $C_{[n]}$ reaches minimum if and only if C' reaches maximum. This reasoning implies that the problem $RDP(1)$ is polynomially equivalent to the Subset Product problem [3] which in the decision version is $\mathcal{NP}$-complete.

Now we present the dynamic programming algorithm DP for solving the $RDP(1)$ problem. The algorithm DP goes through n phases. The kth phase, $1 \leq k \leq n$, produces a set $\mathcal{S}_k$ of states. Any state in $\mathcal{S}_k$ is a vector $S = [s_1, s_2]$. The vector S encodes a partial schedule for the first k jobs under the assumption that $J_{\max}$, by Property 2, is the critical job. The component s_1 of S is the completion time of jobs which have been completed before time $W_{1,1}$ and the component s_2 is the additional total processing time of jobs which start after time $W_{1,2}$.

The sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ are constructed iteratively. An initial set $\mathcal{S}_0$ contains the only state $[t_0, \Delta]$, where $\Delta = W_{1,2} - W_{1,1}$ is the length of the non-availability period. The set $\mathcal{S}_k$ is obtained from the set $\mathcal{S}_{k-1}$ via two mappings, F_1 and F_2, which translate the states of the "old" state space $\mathcal{S}_{k-1}$ into the states of the "new" state space $\mathcal{S}_k$. More precisely, $\mathcal{S}_k = \{F(\alpha_k, S) : S \in \mathcal{S}_{k-1}, F \in \{F_1, F_2\}\}$, where $1 \leq k \leq n$. Intuitively speaking, mapping F_1 "puts" $J_{[k]}$ after time $W_{1,2}$ and mapping F_2 "puts" $J_{[k]}$ before time $W_{1,1}$, if it is possible for the given state. The form of the criterion function $G(S)$ for the problem $RDP(1)$ is given in (1). The algorithm DP is as follows.

Algorithm DP for the problem $RDP(1)$

$\mathcal{S}_0 \leftarrow \{[t_0, \Delta]\}$;
for $k \leftarrow 1$ **to** $n - 1$ **do**
  $\mathcal{S}_k \leftarrow \emptyset$;
  for each $S \in \mathcal{S}_{k-1}$ **do**
    $F_1(\alpha_k, s_1, s_2) \leftarrow [s_1, s_2(1 + \alpha_k)]$;

```
        if s_1(1 + α_k) ≤ W_{1,1} then F_2(α_k, s_1, s_2) ← [s_1(1 + α_k), s_2];
        S_k ← S_k ∪ F_1(α_k, s_1, s_2) ∪ F_2(α_k, s_1, s_2);
    end
end
for each S ∈ S_n do
    G(S) ← τ + s_2;
return min{G(S) : S ∈ S_n}
```

Theorem 1 *For the problem $RDP(1)$ there exists an FPTAS.*

Proof. The mappings F_1 and F_2 are vectors of polynomials with nonnegative coefficients, and the polynomial functions in F_1 and F_2 that yield the first coordinates are polynomials which do not depend on the second coordinate s_2. Moreover, all polynomials linearly depend on s_1 and s_2. The inequality inside operator **if** may be checked in polynomial time, does not depend on s_2 and has the positive coefficient of s_1. Goal function G is a polynomial function with nonnegative coefficients. Moreover, the length of the binary encoding of every value obtained by the algorithm DP is polynomially bounded in the input size, since the length does not exceed $\log((W_{1,2} - W_{1,1}) \prod_{j=1}^{n}(1 + \alpha_j)) \leq n \log \max\{1 + a_{\max}, W_{1,2}\}$. Therefore, the algorithm DP satisfies conditions of Lemma 7.1 and Theorem 3.5 from the paper Woeginger [6], and the problem $RDP(1)$ is a *DP-benevolent* problem. Hence, for the problem $RDP(1)$ there exists an FPTAS. ■

Notice that Property 1 also provides a dynamic programming algorithm for the problem $RDP(h)$ with $h = const$. However, in this case the problem $RDP(h)$ is not a DP-benevolent problem. Moreover, using a simple gap reduction technique we can prove the following result.

Theorem 2 *For the problem $RDP(h)$ with $h \geq 2$, there does not exist a polynomial-time approximation algorithm with a constant worst-case ratio $r > 1$, unless $\mathcal{P} = \mathcal{NP}$.*

Acknowledgement. The research of A. Kononov was supported by RFBR grants 05-01-00075-a and 06-01-00960-a.

References

[1] T-C.E. Cheng, Q. Ding and B.M-T. Lin (2004), A concise survey of scheduling with time-dependent scheduling times, *European Journal of Operational Research* **152**, 1 – 13.

[2] S. Gawiejnowicz (2007), Scheduling deteriorating jobs subject to machine or job availability constraints, *European Journal of Operational Research* **180**, 472 – 478.

[3] M.R. Garey and D.S. Johnson (1979), *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-completeness*, W. H. Freeman, San Francisco, CA.

[4] M. Ji, Y. He and T-C.E. Cheng (2006), Scheduling linear deteriorating jobs with an availability constraint on a single machine, *Theoretical Computer Science* **362**, 115 – 126.

[5] C-Y. Lee (2004), Machine scheduling with availability constraints, Chapter 22 in J.Y-T. Leung (ed.), *Handbook of Scheduling*, Chapman and Hall/CRC, Boca Raton.

[6] G. Woeginger (2000), When does a dynamic programming formulation guarantee the existence of an FPTAS ?, *INFORMS Journal on Computing* **12**, 57 – 73.

[7] C-C. Wu and W-C. Lee (2003), Scheduling linear deteriorating jobs to minimize makespan with an availability constraint on a single machine, *Information Processing Letters* **87**, 89 – 93.

The Parallel Machine Scheduling Problem with Path-like Constraints

Yann Hendel, Wieslaw Kubiak
Memorial University, Canada, Yan.Hendel@gmail.com, WKubiak@mun.ca

1 Presentation of the problem

We are given a set of different jobs $\{J_1, \ldots, J_m\}$. Each job J_i has c_i copies. All these job copies have to be scheduled on F parallel machines. We break with the usual assumption that any subset of jobs can be processed on any machine. Instead, we assume that only some subsets of jobs may be processed on the same machine. These subsets are defined by all s-t paths in an acyclic graph $G = (\{s, t\}, N, A)$, where each arc i represents a job J_i with a processing time p_i and a path from the source s to the sink t of this graph represents a feasible set of jobs that can be executed on the same machine. We search for a schedule, if any exists, which minimizes the makespan.

Our motivation to study this scheduling problem comes from the following network flow context. Given an acyclic directed graph $G = (\{s, t\}, N, A)$, a cost (or *length*) function $p : A \to \mathbb{Z}_{\geq 0}$, a capacity function $c : A \to \mathbb{Z}_{>0}$ and a value F of flow in G. Let $f : A \to \mathbb{Z}_{\geq 0}$ with $f_i \leq c_i$ be a flow in G with its value F. The *decomposition of the flow f into paths* is a multi-set D_f of F paths (in principle, not all different) such that by pushing a unit of flow on each of these paths we obtain the flow f of G. Let $\mathbb{D}_f$ be the set of all the *decompositions of the flow f into paths* and let $D_f \in \mathbb{D}_f$. We denote by μ_π the value of the total flow along the s-t path π in the decomposition D_f. Then, a decomposition D_f is equivalent to a set of pairs (π, μ_π) such that $\sum \mu_\pi = F$. Notice that now the π's are all different. For each path π, we denote by $p(\pi)$ its cost, that is the sum of costs of its arcs. Let $\Lambda = \max_{\pi \in D_f} p(\pi)$. We consider flows f of value F that maximize $\sum_{i \in A} f_i$ and such their decompositions D_f that minimize Λ. We refer to this problem as SLP (for shortest longest path). If the capacity function c obeys the flow conservation law, then there is a flow which saturates every arc. Consequently, the network flow problem becomes the scheduling problem with the number of machines equal the total capacity of the arcs leaving the source s.

This scheduling problem generalizes the well-known identical parallel scheduling to minimize makespan, $P||C_{\max}$. The generalization is shown in Figure 1. The instance of SLP showed in Figure 1 has n jobs with positive processing times, there is a single copy of each of these jobs, and n dummy jobs with negligible processing times, there are $k - 1$ copies of each of these jobs. Moreover, $F = k$. The two numbers on each arc in Figure 1 denote respectively its capacity and its processing time. The flow conservation law is met be the arc capacities. Solving SLP for this instance is equivalent to solving $Pk||C_{\max}$ since an $s - t$ path can take any subset of upper arcs. Therefore, each upper arc must be assigned to a path or equivalently to a machine so as to minimize the makespan.

The SLP is related in the literature to the length-bounded maximum flow problem (see Baier [1] for an overview). In this latter problem, one seeks to maximize the flow with the constraint that there is a decomposition of this flow such that the length of its paths are bounded by a constant. It should be noted that the optimal solution of SLP may not yield a maximal flow.

In this talk, we shall assume that the solution to the scheduling problem exists by assuming that the flow function f is given and $f = c$. It remains to find the decomposition D_f which minimizes Λ (we denote this problem by SLP^f). We address the following cases of this problem: when the value of the flow is bounded by a constant k (that is there are k parallel machines), we denote

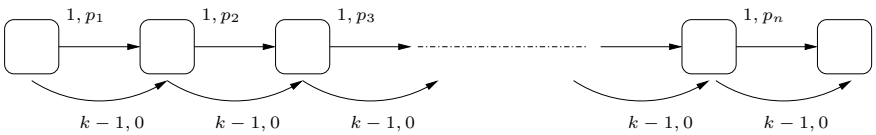


Figure 1: The SLP equivalent to $Pk||C_{\max}$

this problem $\text{SLP}^f_{\text{BFk}}$, when the lengths on the arcs are bounded by a polynomial of the input size, SLP^f_{BC}, and when the flow is unitary on each arc (i.e. there is a unique copy) SLP^f_{UF}.

2 Main results

In the first part of the presentation, we address some computational complexity issues, in particular the question whether the SLP^f problem is in the class NP. We give a short certificate for the case when F is bounded by a polynomial of the input size (the SLP_{BFk} belongs to this case) and therefore this particular case is proven in NP. We then propose some pathologic instances where any flow decomposition of f that minimizes λ in the general SLP^f problem has an exponential number of different paths. These instances provide some evidence that SLP^f, though NP-hard in the strong sense, may not actually be in NP. However, for the case where the cost are bounded, that is ($\text{SLP}^f_{\text{BFk}}$), we provide a second certificate for which the existence of a solution to $\text{SLP}^f_{\text{BFk}}$ can be answered in polynomial time. Thus $\text{SLP}^f_{\text{BFk}}$ is proven in NP.

We then show that SLP^f remains NP-Hard in the strong sense even for SLP^f_{UF}. However, $\text{SLP}^f_{\text{BFk}}$ is proven ordinary NP-hard. Next, we show that $\text{SLP}^f_{\text{BF2}}$ is equivalent to $P2||C_{\max}$ and we propose a pseudopolynomial time dynamic programming algorithm for $\text{SLP}^f_{\text{BFk}}$. This algorithm generalizes the well-known algorithm for $Pk||C_{\max}$ (see [2]). The time complexity of the dynamic programming algorithm is of $O(n\Delta^k)$, where Δ is the length of the longest path in the graph. For the unitary costs, the algorithm runs in polynomial time, its complexity is $O(nm^k)$. Finally, we use this dynamic programming algorithm to develop an FPTAS which runs in $O(\frac{k^k n^{k+1}}{\epsilon^k})$ for any $\epsilon > 0$ for the $\text{SLP}^f_{\text{BFk}}$ problem.

Finally, we present open problems and possible directions for future research.

References

[1] G. Baier (2003). Flows with path restrictions. *PhD thesis, T.U. Berlin*

[2] M.H. Rothkopf (1966). Scheduling independent tasks on parallel processors. *Management Sci.*12, 437-447.

Dynamic Algorithms for Order Acceptance and Capacity Planning within a Multi-Project Environment

Jade Herbots, Willy S. Herroelen, Roel Leus, Erik L. Demeulemeester
Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium, {Jade.Herbots, Willy.Herroelen, Roel.Leus, Erik.Demeulemeester}@econ.kuleuven.be

We consider the dynamic order acceptance and capacity planning problem under limited regular and non-regular resources. The goal consists of maximizing the expected profits of the accepted projects within a finite problem horizon. Capacity planning is a useful tool to support tactical decisions such as due-date quotation, price quotation and hiring non-regular capacity. The way the projects are planned affects their payout time and as a consequence, the reinvestment revenues, as well as the available capacity for future arriving projects. Since actual characteristics of project proposals are only revealed upon arrival, dynamic solution approaches are more likely to obtain good results. For this reason, this paper considers dynamic heuristics, such as approximate dynamic programming algorithms and investigates their suitability to solve the problem. We perform simulation experiments to compare the performance of our algorithms to methods commonly used in practice.

Keywords: approximate dynamic programming, order acceptance, capacity planning, simulation, multi-project.

1 Introduction

Acknowledging the fact that companies have limited resources at their disposal implies that a profit-maximizing company would not accept all project proposals, but would be willing to reject some in order to increase its overall profits. This contrasts sharply with the common practice in project management of accepting all project proposals with a positive net present value (NPV) and to plan them on a first-come, first-served (FCFS) basis, without consideration of future arrivals.

In this paper we examine the order-acceptance and capacity-planning decision facing multi-project organizations upon project arrival. *Capacity planning* determines the allocation of the available (regular and non-regular) resources to the candidate projects, while *order acceptance* is concerned with the accept/reject decision of these projects. In a multi-project environment, projects typically share common resources, so that adequate management of these scarce resources is of crucial importance. Consequently, the development of good acceptance rules and capacity-planning tools is extremely relevant, as they can support decisions such as due-date quotation, price quotation and hiring non-regular capacity. Appropriate order acceptance and capacity planning allows to gain a larger control over the use of non-regular capacity, increase profits and improve delivery performance, which creates a competitive advantage to the company. These benefits constitute the motivation for this research.

Our research adheres to different research domains, one of which is *revenue-based capacity management*, which studies the problem of satisfying customer demand with limited resources while maximizing the company's revenue and profitability [1]. Secondly, the research is related to *portfolio planning and scheduling*, which involves the selection and scheduling/planning of projects. Most of this literature has been dedicated to *static* environments, in which project selection is performed only once, at the beginning of the problem horizon [9], [13]. An example of operational project selection can be found in [15]; within *job-shop planning*, job selection has been a topic of

growing interest in the last decade. In [5] and [12] a number of jobs are considered for selection and subsequently the job sequence is determined for the retained jobs. As for the *dynamic* context, where orders arise dynamically to the organization and require immediate response, the existing work is relatively scarce, although there has been a growing interest in recent years [10], [11]. In [7], *simulation* was used to compare different order-acceptance strategies in a job-shop environment. The same methodology was used in [1] and [14] for production-to-order environments; in addition, heuristics for scheduling the accepted work orders were developed. In a completely different context, a *decision-theory-based approach* was implemented in [2] that reserves parts of the capacity for specified order types through a capacity allocation policy. For a more elaborate survey of the literature on both static and dynamic problems, we refer to [8].

2 Model and solution approach

In this paper, we develop dynamic order acceptance and planning algorithms that aim to maximize the expected profits from accepted orders under finite regular per-period capacity. If needed, non-regular capacity units can be brought in at per-unit costs. Only one resource type is considered, which is taken to represent the *bottleneck* resource of the company, for instance in a manufacture-to-order (MTO) environment it might represent a single machine or a team of engineers. We assume that the company owns a limited number of bottleneck capacity units. The amount of *regular* capacity units is the result of a long-term strategic decision that cannot be revised within the time horizon considered in our planning framework. In contrast, the amount of *non-regular* capacity units can be altered as a result of working overtime, hiring temporary labor or outsourcing.

Upon completion of a project, the project payoff is received; from this point on reinvestment revenues are reaped. The way the projects are planned affects their payout time and as a consequence, the reinvestment revenues, as well as the available capacity for future arriving projects. In our model, each project consists of an aggregated workload on the bottleneck resource, expressed as a discrete number of work packages. Obviously, accepted orders can only be executed between their release time and the project's due date, which is regarded here as a deadline. We assume that the company has forecasts for the main features (workload, pay-off and deadline) of the incoming projects, which are obtained using forecasting techniques.

In [8], we modeled the problem as an extension of the optimal stopping problem, a well-known problem within dynamic programming (DP) [4]. We also presented a stochastic dynamic-programming (SDP) approach that maximizes the expected revenues of the dynamic-order acceptance and capacity-planning problem. Since SDP suffers from Bellman's [3] *curse of dimensionality*, approximate methods are needed to solve real-life problems.

Because actual characteristics of project proposals are only revealed upon arrival, dynamic solution approaches are more likely to obtain good results. For this reason, this paper considers dynamic heuristics in general. More particular, the suitability of approximate dynamic programming algorithms [4] to solve the problem will be investigated. Simulation experiments compare the performance of our procedures to a first-come, first-served policy that is commonly used in practice.

Our algorithms are particularly relevant for environments in which a scarce resource acts as a single static bottleneck and where at least rudimentary information about the work content of the proposed and future projects is available. Examples of such environments are MTOs with a single static bottleneck resource [11], construction environments and maintenance projects [6].

References

[1] C. Akkan (1997), Finite-capacity scheduling-based planning for revenue-based capacity management, *European Journal of Operational Research* **100**, 170 – 179.

[2] N. Balakrishnan, J.W. Patterson and S.V. Sridharan (1996), Rationing capacity between two product classes, *Decision Sciences* **27(2)**, 185 – 214.

[3] R. Bellman (1957), *Dynamic programming*, Princeton University Press, Princeton, NJ.

[4] D.P. Bertsekas (2005), *Dynamic programming and optimal control*, Athena Scientific.

[5] P. De, J.B. Ghosh and C.E. Wells (1993), Job selection and sequencing on a single machine in a random environment, *European Journal of Operational Research* **70**, 425 – 431.

[6] R. De Boer (1998), *Resource-constrained multi-project management - A hierarchical decision support system*, University of Twente, Enschede, the Netherlands.

[7] M.J. Ebben, E.W. Hans and F.M. Olde Weghuis (2005), Workload based order acceptance in job shop environments, *OR Spektrum* **27**, 107 – 122.

[8] J. Herbots, W. Herroelen and R. Leus (2006), Dynamic order acceptance and capacity planning within a multi-project environment, *Technical Report* **KBI 0614**, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Belgium.

[9] H.F. Lewis and S.A. Slotnick (2002), Multi-period job selection: planning work loads to maximize profit, *Computers & Operations Research* **29**, 1081 – 1098.

[10] C.H. Loch and S. Kavadias (2002), Dynamic portfolio selection of NPD programs using marginal returns, *Management Science* **48(10)**, 1227 – 1241.

[11] T.C. Perry and J.C. Hartman (2004), Allocating manufacturing capacity by solving a dynamic, stochastic multiknapsack problem, *Technical Report* **ISE 04T-009**, Lehigh University, PA.

[12] S.A. Slotnick and T.E. Morton (1996), Selecting jobs for a heavily loaded shop with lateness penalties, *Computers & Operations Research* **23**, 131 – 140.

[13] R. Weber, B. Werners and H.J. Zimmerman (1990), Planning models for research and development, *European Journal of Operational Research* **48**, 175 – 188.

[14] F.A. Wester, J. Wijngaard and Z.H. Zijm (1992), Order acceptance strategies in a production-to-order environment with setup times and due-dates, *International Journal of Production Research* **30(6)**, 1313 – 1326.

[15] K.-K. Yang and C.C. Sum (1997), An evaluation of due date, resource allocation, project release, and activity scheduling rules in a multi-project environment, *European Journal of Operational Research* **103**, 139 – 154.

Exact and Suboptimal Reactive Strategies for Resource-Constrained Project Scheduling with Uncertain Resource Availabilities

Olivier Lambrechts, Erik L. Demeulemeester, Willy S. Herroelen
Katholieke Universiteit Leuven, Department of Decision Sciences and Information Management, Belgium,
{Olivier.Lambrechts, Erik.Demeulemeester, Willy.Herroelen}@econ.kuleuven.be

In order to cope with the uncertainty inherent in practical project management various strategies can be used. Proactive strategies try to accommodate disruptions in advance, whereas reactive strategies react after a disruption happened and try to revert to a feasible schedule. We give an extensive overview of reactive strategies, exact as well as heuristic, that can be used during project execution when the project is subject to disruptions due to unforeseen resource breakdowns. Furthermore, we present a heuristic that also takes future uncertainty into account when repairing the schedule.

1 Introduction

Traditional scheduling methods have only focused on deterministic environments in which all information is given in advance and is not subject to change. However, this will seldom be the case in practice. Delays may be caused by bad weather conditions, resource failures, absenteeism, activity duration increases, etc. Proactive strategies try to accommodate these disruptions in advance in order to minimize the negative impact of activity starting time delays. Unfortunately, totally eliminating their occurrence is economically unviable. One therefore also has to resort to good rescheduling techniques enabling the project manager to restore feasibility while incurring an instability penalty that is as small as possible.

The objective of the proactive-reactive project scheduling problem is to minimize schedule nervousness while meeting precedence, resource and due date constraints. This objective is measured by the sum of the weighted deviations between the original baseline schedule that is constructed before project execution starts and the expected actually realized schedule. The realized activity starting times are stochastic variables that depend on the baseline schedule (which we assume given), on the disturbances encountered during project execution and on the reactive strategy that is used to restore feasibility.

In case an infeasibility occurs due to a resource breakdown, schedule feasibility needs to be restored by postponing one or more of the offending activities in progress on the resource type causing the infeasibility during the period the disruption occurs. Our global objective is to minimize schedule instability. In case the encountered disruption is the last disruption until project completion, the optimal policy will be to create a feasible schedule for which the weighted deviation from the preschedule is as small as possible. This problem is studied in section 2. However, in practice we will usually continue facing resource breakdowns. Therefore, we want to find a schedule that is feasible, does not deviate too much from the original baseline schedule and is well protected against the occurrence of future disruptions. This problem is studied in section 3.

2 Reactive procedures

2.1 Exact approach

First of all, an exact approach has been developed that reduces the problem by creating a new scheduling problem for each preemption alternative. A preemption alternative is defined as a subset of the activities in progress during the time period the disruption occurs and that resolves the infeasibility when all of the activities contained in the alternative are preempted and postponed for at least one time period. The procedure iterates over all these preemption alternatives, creates a reduced problem instance for each alternative and then optimally solves the rescheduling problem corresponding to this reduced problem using an exact approach for solving the resource-constrained project scheduling problem with weighted earliness and tardiness costs that was developed by Vanhoucke et al. [1]. The best solution over all preemption alternatives is then the optimal solution for the rescheduling problem at hand.

2.2 Heuristic procedures

Inspired by the promising results of the use of priority lists in machine and project scheduling, we propose to use a simple reactive strategy relying on *list scheduling*. First of all, a *random* precedence feasible priority list is included for benchmarking purposes. However, we expect far better results from a *scheduled order list* that allows us to reschedule the activities in the order dictated by the baseline schedule (the lowest activity number being the tie-breaker). This priority list is then decoded into a feasible schedule using a *modified serial schedule generation scheme* that takes the known resource availabilities up to the current time period into account.

The *scheduled order list* approach is able to very quickly generate feasible solutions with a reasonable quality. However, solutions may be improved by superimposing a tabu search based improvement heuristic on the priority list rule. This procedure will try to improve the starting solution by iteratively executing the best precedence feasible interchange of two activities in the priority list that does not lead to a state included in the tabu list. The objective is to find a precedence feasible ordering of activities corresponding to a feasible schedule that deviates as little as possible from the baseline schedule S^0.

2.3 Hybrid procedure

In practice, a project manager will spend less time and effort on small disruptions than on disturbances having a major impact on project stability. Therefore, we present a new approach combining elements from both the exact and the heuristic solution procedures. Whenever an infeasibility occurs, a repaired schedule is quickly generated using the 'scheduled order' list-based heuristic. The weighted instability cost of this new, repaired schedule is then compared to the instability cost of the previous, but now infeasible schedule. In case the relative difference is larger than a preset cutoff percentage, we repair the schedule using the exact procedure. If not, the scheduled order schedule is retained.

3 Rescheduling for stability and robustness

The approaches we studied up to now were myopic strategies insofar that they try to optimize the global objective by locally minimizing the difference between the baseline schedule and the repaired schedule. However, it seems naive to assume that schedule uncertainty ceases to exist after the current disruption. Therefore, it is worthwhile to develop a rescheduling approach that does not

only look backwards in time but also adequately tries to protect the schedule from disruptions that might still occur at some future point in time. A metaheuristic was developed that optimizes the bi-objective problem of weighted deviation minimization combined with robustness maximization. Because of the computational problems involved in analytically determining robustness we use a surrogate objective based on the expected activity duration increases due to breakdowns under various assumptions.

4 Computational Experiment

In order to test the relative performance of our reactive strategies we set up a computational experiment using the 480 30-activity test instances contained in the well-known PSPLIB set of project network instances [2]. For each instance a number of disruption scenarios and baseline schedules were considered. For more information regarding the construction of robust project baseline schedules when faced with resource breakdowns we would like to refer to [3] and [4]. Resource breakdowns were modeled using exponential interfailure and repair times.

It can be observed that scheduled order list scheduling performs quite well given the time necessary to execute the algorithm. However, even when only allowing for adjacent interchanges and 50 iterations, tabu search outperforms random as well as scheduled order list scheduling. These results can even be improved by allowing for general interchanges and more iterations. Surprisingly, when allowing for 200 iterations, the tabu search procedure even sometimes outperforms the exact approach. This is probably due to truncating the exact approach after a certain period of time.

As expected, the results of the hybrid procedure lie between those of optimal rescheduling and those of the scheduled order priority list. With only a small increase in required computation time, the procedure is able to yield significantly better results than the simple list scheduling heuristic. However, tabu search based improvement of the scheduled order heuristic still seems to be the most attractive option.

Reactive procedures incorporating robustness always perform worse than a pure instability-based strategy such as the tabu search procedure. This is no doubt due to the fact that we only penalize deviation from the starting schedule. Things change considerably, however, if we do not only consider instability performance but also penalize the number of rescheduling actions. The attractiveness of rescheduling for robustness depends on the ratio of the instability over the rescheduling costs.

References

[1] M. Vanhoucke, E. Demeulemeester and W. Herroelen (2001), An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problems, *Annals of Operations Research* **102**, 179 – 196.

[2] R. Kolisch and A. Sprecher (1997), PSPLIB - A project scheduling library, *European Journal of Operational Research* **96**, 205 – 216.

[3] O. Lambrechts, E. Demeulemeester and W. Herroelen (2007a), Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities, *Journal of Scheduling, to appear.*

[4] O. Lambrechts, E. Demeulemeester and W. Herroelen (2007b), A tabu search procedure for developing robust predictive project schedules, *International Journal of Production Economics, to appear.*

Integrated Vehicle and Crew Scheduling for Extra-Urban Transport

Benoît Laurent
PERINFO, 41 avenue Jean Jaures, France, blaurent@perinfo.com
Jin-Kao Hao
LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers Cedex 01, France, hao@info.univ-angers.fr

1 Introduction

Crews and vehicles are the main resources to provide services in transport systems. The way these resources are employed directly impacts the quality of service and the operational costs, which explains the importance gained by transportation scheduling systems.

The conventional crew and vehicle scheduling process is the *sequential* approach which determines first the vehicles schedule and then the crews schedule. This dichotomy is mainly due to the complexity of each sub-problem. Indeed, the Multi Depot Vehicle Scheduling Problem (MDVSP) is known to be NP-hard ([2]) and the Bus Driver Scheduling Problem (BDSP) also constitutes a difficult problem.

In the early 1980s, Ball et al. criticized this sequential approach [1], but the first real *integrated* solutions, in which vehicles and crews are simultaneously scheduled, were only developed in 1995 ([3]). This integrated treatment of both resources demonstrated its efficiency in [4] where relief locations, i.e. places where a driver can be relieved by a colleague, are spatially distant. Integration is also profitable when a driver is not allowed to change from one vehicle to another.

In this study, we propose a new heuristic approach based on a constraint satisfaction optimization problem for the simultaneous vehicle and crew scheduling problem. We implemented a Greedy Randomized Adaptive Search Procedure (GRASP) which constitutes, to our knowledge, the first application of metaheuristics to this problem. In our situation, all vehicles are parked within the same depot. However, the problem is more general than the usual single depot case which imposes a homogenous fleet of vehicles. Here, they may belong to different categories. Treating an extra-urban situation, we adopt the same assumption as in [4], namely that changeovers only take place at the depot.

2 Vehicle and Crew Scheduling: Problem Presentation

Informally, given a set of trips within a one-day planning horizon, a set of drivers, a fleet of vehicles parked at a given depot, a set of workday types, the Vehicle and Crew Scheduling Problem (VCSP) consists in determining a minimum cost schedule for crews and vehicles, such that generated duties are feasible and mutually compatible.

The trips are characterized by starting and ending locations with corresponding times. Travel times between all pairs of locations are also known. The other inputs concern the vehicles availability per category. Similar bounds exist for the crews.

The whole schedule has to comply with a set of imperative constraints. Crews and vehicles assigned to successive trips must have enough time between these trips to move from one to the other. Category requirements must be satisfied as well. Eventually, crew members are subjected to labor rules (maximum length, maximum working time, etc.).

The cost structure of the problem is composed of fixed and operational costs. For evident economic reasons, the main objective is to reduce the number of working drivers and used vehicles. In order to further reduce costs, it is also useful to minimize idle time and deadheads.

3 Problem Formulation and Solution Approach

Our problem is modeled as a constraint satisfaction optimization problem inspired from [6]. We aim to simultaneously assign a couple (driver and vehicle) to each trip. Therefore, we will define the set of decision variables as the set of trips $\mathcal{T}$. Naturally, the associated value domain $\mathcal{I}_k$ for each such variable corresponds to driver-vehicle pairs.

To tackle this problem, we implemented a Greedy Randomized Adaptive Search Procedure. GRASP is a multi-start metaheuristic for combinatorial problems, in which each iteration consists basically of two phases: construction and local search. Constraint programming techniques are used to build initial solutions. Improvements of these solutions are achieved with a local search algorithm which embeds a powerful "ejection chain" neighborhood exploration mechanism.

4 Comparison Between Sequential and Integrated Scheduling

In our computational experiments, we compared the sequential and the integrated approach on 7 real-world instances representing different workloads. For each approach, we report in Table 1 the number of required drivers and vehicles, the total duration of deadheads (dh) and idle periods (idle). For the sequential approach, we developed a Branch & Bound algorithm on a multi-commodity flow model for the vehicle part and on a set covering model for the crew part (see [5]). Concerning the tuning of GRASP, after some preliminary experimentations, we obtained the best results with the following set of parameters:

- 10 iterations,
- a size of 5 for the restricted candidate list,
- each local search procedure stops after $100 \times |\mathcal{T}|$ iterations without improvement.

With these settings, the computational effort does not exceed 10 minutes on the largest instances. Figures for the simultaneous approach correspond to the best solution obtained over 20 runs.

Table 1: Comparison between sequential and integrated scheduling

	Seq				Int. GRASP			
	drivers	vehicles	dh	idle	drivers	vehicles	dh	idle
bea_59	18	16	23:17	3:54	16 (16.0, 0.0)	16 (16.0, 0.0)	24:24	4:14
cor_67	20	15	19:28	1:55	15 (15.8, 0.4)	15 (15.0, 0.0)	22:12	0:28
cha_105	22	22	39:38	9:36	22 (22.0, 0.0)	22 (22.0, 0.0)	40:53	9:44
sem_151	27	27	57:05	15:45	27 (27.0, 0.0)	27 (27.0, 0.0)	58:20	15:07
dij_159	34	29	51:56	16:02	29 (29.0, 0.0)	29 (29.0, 0.0)	61:10	14:29
otp_215	-	48	-	-	49 (49.0, 0.0)	49 (49.0, 0.0)	145:17	21:13
aux_249	48	44	115:35	22:59	46 (46.8, 0.4)	44 (44.0, 0.0)	123:35	21:21

From this Table, we observe that the integrated approach clearly outperforms the sequential one. In particular, the savings in terms of number of drivers are significant. The sequential approach

provides a lower bound in the number of vehicles that is always reached in the integrated solutions. The only loss concerns the dh-tasks: the schedules are sub-optimal regarding this criterion to allow drivers reliefs.

The integrated approach is more powerful than the sequential one. The latter sometimes fails to solve instances - it occurs with "opt_215" - while solutions are possible when crews and vehicles are considered simultaneously. In the sequential methodology, the vehicle phase might result in a schedule with no relief opportunity along some bus duty and consequently leads to an unresolvable problem for the driver scheduling part.

5 Conclusion

We proposed a new heuristic on a simultaneous drivers and vehicles scheduling problem in an extra-urban area. The assumptions retained are suitable to tackle practical problems in this context. The ability of managing a heterogeneous fleet for a given depot is thus especially relevant.

The formulation as a constraint satisfaction optimization problem and the application of meta-heuristics on such a problem constitute to our knowledge the first attempts in the targeted field.

The computational study carried out on a set of real-world instances clearly shows the dominance of the integrated approach over the conventional sequential approach.

References

[1] M. Ball, L. Bodin, and R. Dial (1983), A matching based heuristic for scheduling mass transit crews and vehicles, *Transportation Science* **17**, 4 – 31.

[2] A. Bertossi, P. Carraresi, and G. Gallo (1987), On some matching problems arising in vehicle scheduling models, *Networks* **17**, 271 – 281.

[3] R. Freling, G. Boender, and J. M. P. Paixão (1995), An integrated approach to vehicle and crew scheduling, Technical Report 9503/A, Economie Institute, Erasmus University Rotterdam, Rotterdam.

[4] A. Gaffi and M. Nonato (1999), An integrated approach to the extra-urban crew and vehicle scheduling problem, In N. H. M. Wilson, editor, *Computer-Aided Transit Scheduling*, Berlin, Springer Verlag, 103 – 128.

[5] D. Huisman (2004), *Integrated and Dynamic Vehicle and Crew Scheduling*, PhD thesis, Tinbergen Institute, Erasmus University Rotterdam.

[6] B. Laurent and J.-K. Hao (2007), Simultaneous vehicle and driver scheduling: a case study in a limousine rental company, to appear in *Computers & Industrial Engineering*.

Scheduling Problems of Chemical Experiments

Vassilissa Lehoux-Lebacque, Nadia Brauner, Gerd Finke, Christophe Rapine
G-SCOP Laboratory, 46 av. Félix Viallet, France,
{vassilissa.lebacque, nadia.brauner, gerd.finke, christophe.rapine}@g-scop.inpg.fr

The scheduling problem we describe was proposed by the Institut Français du Pétrole (IFP) and was studied in collaboration with the two laboratories Gilco-INPG and Leibniz-IMAG in Grenoble. We developed a software to solve efficiently the industrial problem and derived several interesting theoretical problems that are presented in this paper.

1 Description of the problem

We have to schedule chemical experiments with resource constraints. The aim of the experiments is to find the best conditions to perform the synthesis of certain chemicals. Each task is the set of all the experiments that will be conducted for a given chemical. This set consists of two series of experiments: stage 1 and stage 2. The aim of the first stage is to find the best heating duration for the chemical. We start with a given heating time. If it was too short, the experiment is started again with an increased heating duration, and if it was too long, with a decreased duration. Once the perfect heating time is found, we can start the other stage. There, we modify slightly the conditions of the experiments with the heating time obtained in stage 1 as to improve the quality of the chemical obtained. For this stage, the experiments are independent and can be performed in parallel.

To conduct these experiments, we use a set of cyclic containers in which we can perform several experiments at the same time, provided that they have exactly the same duration. Each experiment lasts between 3 and 21 days, and the total number of experiments is several thousands.

The purpose of the software we developed was both maximizing the filling rate of the containers in order to finish as early as possible the set of all the tasks and terminating evenly the tasks as to regularly obtain results on the quality of the chemicals.

The operator is needed at the beginning and at the end of each heating operation, but not during its processing. The planning must take into account the days-off, weekends, etc.

From this industrial problem, we developed several research directions that are presented in the following sections.

2 Considering periods where the tasks can neither start nor end

In the classical scheduling literature, the notion of machine unavailability periods is well known (for maintenance, for example). In our case, we have special periods (weekends, holidays, etc.) where the chemists are not available. However, the presence of the chemists is required to handle the starting and termination of the experiments. Therefore, the subsequent type of problems occurs: we have parallel machines, with a given set of operator-non-availability periods $S(I)$ described as open intervals. The solutions of these problems are schedules with no operation starting nor terminating in any interval of $S(I)$.

The problem of minimizing the makespan is NP-hard and this is already true on one machine with only one interval and independent operations. The problem may be formulated as follows. Given are n operations with processing times $p_1, p_2, \ldots, p_n$ and a single operator-non-availability

period (o-na-period) of duration Δ located at $[C, C+\Delta]$. We may distinguish between three classes of these problems:

(1) all processing times are smaller than Δ: $p_i < \Delta \quad \forall i$;

(2) there are processing times smaller than Δ and others greater than Δ;

(3) all processing times are greater than Δ: $p_i \geq \Delta \quad \forall i$.

In particular, case (3) applies to our applications. The length of the experiments are given in days (≥ 3) and the o-na-periods are usually 2 days (weekends). In this case, we prove that on one machine and one interval, makespan minimization can be done in $O(n \log n)$ where n is the number of experiments.

3 Scheduling chains of operations on a batching machine with task compatibility

At an intermediate step of stage 1, we are confronted with the scheduling of chains of operations which come from the different chemicals. Therefore in the classical notation of scheduling, we have tasks where the precedence graph is a chain. In addition, our machines are so-called *batch machines*, *i.e.*, machines that can handle simultaneously several operations [1]. We call two operations *batch compatible* if they have exactly the same length which is then also the length of the batch. In the scheduling context, we have problems of scheduling on batch machines with precedence constraints and batch compatibility. The objective is to minimize the makespan, *i.e.*, to finish as quickly as possible all operations.

The following example is composed of two tasks with respective sequences of operations

$$1 \rightarrow 21 \rightarrow 5 \rightarrow 14 \quad \text{and} \quad 1 \rightarrow 5 \rightarrow 14 \rightarrow 21$$

A possible schedule of length 61 is:

1	21	5	14	21
1		5	14	

where the operations of length 1, 5 and 14 are grouped in batches of capacity 2 and the operations of length 21 are scheduled alone. An optimal schedule of length 60 is:

1	5	14	21	5	14
1			21		

This problem contains as a subcase the shortest subsequence problem which is already NP-hard. Interesting new problems and results where derived from this study with a connection to sequence alignment in bio-informatics.

4 Minimizing the total completion time of the chemicals

At the final stage of the experiments, one has to schedule tasks that are all in stage 2. So for each task, we have a set of operations to finish as fast as possible (this would be in fact, the finishing of one chemical). To state on the interest of industrial production for a given chemical, the chemists need the results of all its experiments. We are therefore focussing on the end of the last experiment of each chemical and not that of each of its operation. Since we want to finish stage 2 for all given chemicals, the problem can be stated as follows: a task T_i is a set of operations and C'_i is the completion time of task T_i, *i.e.* the end its last operation. All tasks are independent and can

be processed in parallel. The objective is to minimize $\sum C'_i$. Observe that this problem is closely related to [3]. Again we have to schedule such tasks on a system of parallel machines. Note that this problem is different from the classical scheduling problem $Pm||\sum C_i$ where C_i is the termination of each single operation without the set structure explained above. In the classical case, SPT gives an optimal solution in polynomial time [4]. The problem is not either directly related to $Pm||\sum w_i C_i$ since you cannot know which will be the last operation of a task.

Returning to our case, the problem is easy if restricted to a single machine: SPT applied to the sum of the durations of operations for the tasks is optimal. Therefore, $1||\sum C'_i$ is polynomially solvable.

For two-machines ($P2||\sum C'_i$), the problem is already NP-hard: it contains the minimization of the makespan ($P2||Cmax$) as a subcase, which is known to be NP-hard [2] (consider a single task).

References

[1] M. Boudhar and G. Finke (2000), Scheduling on a Batch Machine with Job Compatibilities., *Belgian Journal of Operations Research, Statistics and Computer Science - JORBEL* **40**, 69 – 80.

[2] M.R. Garey and D.S.Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, CA.

[3] J.Y-T. Leung, H. Li and Pinedo (2005), Order scheduling models: an overview. In *First Multi-disciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003)*, volume 1, 37 – 53, Nottingham, UK, 2003.

[4] M. Pinedo (1995), *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, NJ.

Project Scheduling with Success or Failure in Individual Activities

Roel Leus
KULeuven, Department of Decision Sciences and Information Management, Naamsestraat 69, 3000 Leuven, Belgium, Roel.Leus@econ.kuleuven.be

Bert De Reyck
London Business School, Regent's Park, London NW1 4SA, United Kingdom, bdereyck@london.edu

We examine how to schedule projects in order to maximize their expected net present value when the project activities have a probability of failure and when an activity's failure leads to overall project termination. We formulate the problem, show that it is NP-hard, develop a branch-and-bound algorithm that allows to obtain optimal solutions and provide extensive computational results.

1 Introduction

An important feature of Research-and-Development (R&D) projects is that, apart from the commercial and market risks common to all projects, their constituent activities also carry the risk of technical failure. Therefore, besides projects overrunning their budgets or deadlines and the commercial returns not meeting their targets, R&D projects also carry the risk of failing altogether, resulting in time and resources spent without any tangible return. In this paper, we tackle the problem of scheduling the activities of an R&D project that is subject to technological uncertainty, i.e. in which the individual activities carry a risk of failure, and where an activity's failure results in the project's overall failure. The goal is to schedule the activities in such a way as to maximize the expected net present value of the project, taking into account the activity costs, the cash flows generated by a successful project, the activity durations and the probability of failure of each of the activities.

The model developed in this paper is useful for any R&D setting where activities carry a risk of failure, and is of particular interest to drug-development projects in the pharmaceutical industry, in which stringent scientific procedures have to be followed to ensure patient safety in distinct stages before a medicine can be approved for production. The project may need to be terminated in any of these stages, either because the product is revealed not to have the desired properties or because of harmful side effects. The failure of one of the stages results in overall project termination. As stated by [6], "If a drug candidate fails during the development phase it is withdrawn entirely from further testing. Unlike in the automobile industry, drugs are not modular products where a faulty stick shift can be replaced without throwing the entire car design away. In pharmaceutical R&D, drug design cannot be changed."

2 Problem formulation

We wish to maximize the expected net present value (NPV) of a project by constructing a project schedule specifying when to execute each activity. The final project payoff is only achieved when all activities are successful, and the project is terminated as soon as an activity fails. We focus on the case where all activity cash flows during the development phase are negative, which is typical for R&D projects. Activity success or failure is revealed at the end of each activity. Consequently, each activity will only be started if all the activities scheduled to finish earlier have a positive outcome. Therefore, in the objective function, the activity cash flows are weighted by the probability of joint success of all its scheduled predecessors. We make abstraction of resource constraints and

N	$= \{0, 1, \ldots, n\}$, the set of project activities; $N_i = N \backslash \{i\}$ $(i \in N)$ and $N_{0n} = N \backslash \{0, n\}$
c_i	cash flow of activity $i \in N_n$, non-positive integer; incurred at the start of the activity
C	integer end-of-project payoff, ≥ 0; received at the start of activity n
d_i	duration of activity $i \in N_n$, non-negative integer (positive for $i \in N_{0n}$)
p_i	probability of technical success (PTS) of activity $i \in N_n$
r	continuous discount rate
A	(strict) partial order on N, i.e. an irreflexive and transitive relation, representing technological precedence constraints
s_i	starting time of activity $i \in N$, ≥ 0; starting-time vector $\mathbf{s}$ is a schedule
δ	project deadline

Table 1: Definitions.

duration uncertainty, and consider the success probabilities of the different tasks as independent. The parameters that are used throughout the paper are defined in Table 1.

Without loss of generality, we assume activity 0 to be a dummy representing project initiation, with $c_0 = d_0 = 0$ and $p_0 = 1$, and $(0, i) \in A$ for all $i \in N_0$. Activity n represents project completion and is a successor of all other activities. Activities N_{0n} are referred to as *intermediate* activities; we assume that $d_i > 0$ for $i \in N_{0n}$. A deadline δ on the schedule length is imposed: we require that $s_n \leq \delta$. This deadline is needed because optimization will try to push activity start times to infinity if the optimal expected NPV of a particular problem instance is negative. A second reason for using a deadline is that it allows to examine the impact of schedule length on the quality of the schedule.

In order to formulate the problem we wish to solve, we define the additional variables

$$q_i(\mathbf{s}) = \prod_{\substack{k \in N: \\ s_k + d_k \leq s_i}} p_k$$

associated with activities $i \in N_0$. Remark that $q_n(\mathbf{s})$ is a constant, independent of the schedule; we write $q_n \equiv q_n(\mathbf{s})$. q_i represents the probability that activity i is executed, and thus needs to be paid for. We now formally state the *R&D-Project Scheduling Problem* or RDPSP:

$$\max \qquad g(\mathbf{s}) = q_n C e^{-rs_n} + \sum_{i=1}^{n-1} q_i(\mathbf{s}) c_i e^{-rs_i} \tag{1}$$

subject to

$$\begin{aligned} s_i + d_i &\leq s_j && \forall (i, j) \in A \\ s_n &\leq \delta \\ s_i &\geq 0 && \forall i \in N \end{aligned}$$

In the objective function $g()$, each activity cash flow c_i is weighted with two factors, namely with $q_i(\mathbf{s})$, the probability of joint success of all predecessors in time, and with a discount factor e^{-rs_i}, dependent on the starting time s_i of activity i.

3 Properties

Theorem 1. *If $r = 0$ and $\delta \geq \sum_{i \in N_n} d_i$ then an optimal feasible schedule exists without activities scheduled in parallel.*

The proofs of the theorems appear in [4]. Intuitively, the theorem says that when money has no time value, it is a dominant choice to perform all tasks sequentially. Theorem 1 allows us to establish ties with the literature on sequential testing. We define problem LCT ('*least-cost testing*') as problem RDPSP whose solution space is restricted to schedules that impose a complete order on N; remark that LCT is not a sub-problem of RDPSP since we restrict the set of solutions and not the input parameters.

Without dummy start and end (and so without final project payoff), a number of special cases of LCT with $r = 0$ can be solved in polynomial time. If $A = \varnothing$ then each schedule that sequences the activities in non-increasing order of $c_i/(1 - p_i)$ is optimal. One of the earliest references for this result seems to be [8]; another source is [2]. A polynomial-time algorithm for LCT also exists when $G(N, A)$ consists of a number of *parallel chains* (see [3]). Based on [9] it can be shown that the problem is also solvable in polynomial time when $G(N, A)$ is *series-parallel*.

The foregoing results carry over to RDPSP when $\delta \geq \sum_{i \in N_n} d_i$ and $r = 0$. However, the incorporation of precedence constraints taking the form of an arbitrary acyclic digraph $G(N, A)$ results in an NP-hard problem:

Theorem 2. RDPSP *is* NP*-hard in the ordinary sense, even if* $r = 0$, $C = 0$, $\forall i \in N_{0n} : d_i = 1$, *and* $\delta \geq \sum_{i \in N_n} d_i$.

Corollary 1. LCT *is ordinarily* NP*-hard under the same conditions.*

This corollary settles what is said to be an open problem in [9] and [11].

4 A branch-and-bound algorithm

For an arbitrary relation E on N, define $\mathcal{S}(E) = \{\mathbf{s} \in \mathbb{R}_{\geq}^{n+1} : s_i + d_i \leq s_j, \forall (i, j) \in E\}$, which is a convex polyhedron ($\mathbb{R}_{\geq}$ denotes the set of positive real numbers). $\mathcal{S}(E)$ is non-empty if and only if the corresponding precedence graph $G(N, E)$ is acyclic. The set of feasible schedules for RDPSP is $\{\mathbf{s} \in \mathcal{S}(A) : s_n \leq \delta\}$. Clearly, if $A \subseteq E$ then $\mathcal{S}(E) \subseteq \mathcal{S}(A)$. If $A \subseteq E$ and $G(N, E)$ is acyclic, we say that E is a *feasible extension* of A. For a given schedule $\mathbf{s}$, we define the schedule-induced strict order $R(\mathbf{s}) = \{(i, j) \in N \times N | i \neq j \wedge s_i + d_i \leq s_j\}$, which corresponds to the precedence constraints implied by $\mathbf{s}$ (see e.g. [1, 10]).

RDPSP is solved in two phases. In the first phase we produce a feasible extension E of A, which yields values

$$y_i(E) = \prod_{(k,i) \in E} p_k$$

for activities $i \in N_{0n}$. We then substitute values $y_i(E)$ for $q_i(\mathbf{s})$ in the objective function (1) for each $i \in N_{0n}$, and optimize $g(\mathbf{s})$ subject to the constraints that $\mathbf{s} \in \mathcal{S}(E)$ and $s_n \leq \delta$. If we implicitly or explicitly enumerate all feasible extensions E of A, we are guaranteed to find an optimal schedule for RDPSP, since for each feasible schedule $\mathbf{s} \in \mathcal{S}(A)$ it holds that $\mathbf{s} \in \mathcal{S}(R(\mathbf{s}))$, and $R(\mathbf{s})$ extends A; a corresponding relation E is called an *optimal feasible extension*. This enumeration process is embedded into a branching procedure, which, in combination with upper bounds on the best objective-function value reachable from a given node in the resulting search tree, leads to a branch-and-bound (B&B) procedure that allows us to find optimal solutions.

The second phase (optimization after substitution of the values y_i), to be examined for each feasible extension E, amounts to project scheduling with NPV objective without resource constraints (see [7]). In this case, the scheduling problem is easily solved because all intermediate cash flows are non-positive: each activity can be scheduled to end at the earliest of the starting times of its

successors in E. Depending on whether the corresponding expected NPV is positive or negative, we set $s_0 = 0$ or $s_n = \delta$, respectively.

In our presentation we will report on computational results for the B&B-algorithm on a number of sets of test instances generated by RanGen [5].

References

[1] M. Bartusch, R.H. Möhring, F.J. Radermacher (1988), Scheduling project networks with resource constraints and time windows, *Annals of Operations Research* **16**, 201 – 240.

[2] R. Butterworth (1972), Some reliability fault-testing models, *Operations Research* **20**, 335 – 343.

[3] S.Y. Chiu, L.A. Cox Jr., X. Sun (1999), Optimal sequential inspections of reliability systems subject to parallel-chain precedence constraints, *Discrete Applied Mathematics* **96–97**, 327 – 336.

[4] P. Crama, B. De Reyck, Z. Degraeve, R. Leus (2005), Managing technology risk in R&D project planning: Optimal timing and parallelization of R&D activities, Research Report 0501, Department of Applied Economics, KULeuven.

[5] E. Demeulemeester, M. Vanhoucke, W. Herroelen (2003), A random generator for activity-on-the-node networks, *Journal of Scheduling* **6**, 13 – 34.

[6] O. Gassmann, G. Reepmeyer, M. von Zedtwitz (2004), *Leading Pharmaceutical Innovation, Trends and Drivers for Growth in the Pharmaceutical Industry,* Springer-Verlag, Berlin Heidelberg New York.

[7] W.S. Herroelen, P. Van Dommelen, E.L. Demeulemeester (1997), Project network models with discounted cash flows: a guided tour through recent developments, *European Journal of Operational Research* **100**, 97 – 121.

[8] L.G. Mitten (1960), An analytic solution to the least cost testing sequence problem, *Journal of Industrial Engineering* **11**, 17 – 17.

[9] C.L. Monma, J.B. Sidney (1979), Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research* **4**, 215 – 224.

[10] K. Neumann, C. Schwindt, J. Zimmermann (2003), *Project Scheduling with Time Windows and Scarce Resources: Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions.* Springer-Verlag, 2nd edition.

[11] T. Ünlüyurt (2004), Sequential testing of complex systems: a review, *Discrete Applied Mathematics* **142**, 189 – 205.

A Branch-and-Price Procedure for Nurse Staffing Incorporating Roster Preferences

Broos Maenhout
Ghent University, Hoveniersberg 24, B-9000 Gent, Belgium, Broos.Maenhout@Ugent.be
Mario Vanhoucke
Ghent University, Hoveniersberg 24, B-9000 Gent, Belgium, Mario.Vanhoucke@Ugent.be
Vlerick Leuven Gent Management School, Reep 1, B-9000 Gent, Belgium, Mario.Vanhoucke@Vlerick.be

1. Introduction

Managing, recruiting, keeping the right staff, and deploying resources efficiently are key challenges for the healthcare industry in the coming decennia due to the increasing elder population and the corresponding increasing demand for care and caring personnel ([16]). Since the quality of care ultimately depends on the quality and motivation of health human resources, personnel problems should be appropriately addressed since staff shortages or unmotivated health workforce are likely to have adverse effects on the delivery of health services and outcome of care. A key factor is the organizational support to employees which is especially revealed in the personnel staffing policies and scheduling practices conducted by the health organizations. On the one hand, it is of critical importance to deploy efficiently the available personnel since this is a large determinant of service organization efficiency and customers' requirements satisfaction in providing the continuity of care. On the other hand, [7] identify unattractive schedules and high workloads as two important factors leading to discontentment and a high nursing turnover.

In order to optimize the organizational support to the largest extent, we integrate the staffing and scheduling phase of the nurse workforce management process ([1], [3], [13], and [14]). The staffing phase contains a strategic budgeting decision which boils down to determining the number of nurse labour hours required of each nursing skill class to meet the forecasted patient demand. Based on these authorized budgets for full-time equivalent nursing personnel and the allowable usage of supplemental staffing resources, the number of permanent nursing positions of each type (both full- and part-time) is established within budgetary constraints. The scheduling phase focuses on the assignment of nurses to work days and/or daily work shifts across a typical planning horizon of 1 to 8 weeks for each nursing unit in order to satisfy the minimal coverage requirements while meeting legal, union, hospital and personal constraints imposed on the nurses' individual schedules. An overview of the extensive literature on personnel scheduling can be found back in [4] and [10]. Personnel scheduling solution procedures are already applied to real-world problems (e.g., in [5]). Despite the wide impact of the staffing decisions on the alternatives in the scheduling phase, these two phases have been mainly investigated separately in the literature. Little research has been carried out that integrate staffing and scheduling decisions (e.g., [9], [13], and [15]).

2. Problem description

In hospitals nurses are typically assigned to a ward. This assignment is done rather arbitrarily by the department head based on the nurse's competencies and the nurse's ward preference. Next to being assigned to a specific ward, nurses can be assigned to float between different wards which is nowadays one of the most common staffing strategies ([11]). Advantages and disadvantages of this staffing policy are provided in the literature survey of [8]. The use of a float pool is typically a cost-saving and efficient staffing strategy which diminishes inadequate staffing levels, eliminates unneeded manpower and reduces the number of budgeted hours, overtime hours and/or the need for reliance on costly agency or per diem staff. However, this strategy is generally recognized as to be at the expense of the nurse (dissatisfaction, stress, poor group dynamics, etc) and the patient's

quality of care (i.e., floating staff needs to be competent and familiar with the patient care practices). Evidently, the health organization needs to offer something in exchange for this flexibility to compromise and motivate nurses to guarantee the provided quality of care. Typically, nurses need to be (socially) satisfied by catering for their individual roster preferences as much as possible. In this respect, the interaction between the staffing and the scheduling phase becomes interesting. Modelling, quantifying and parametrizing these considerations, nurses are assigned to one of the wards or to the float pool and rostered with a certain degree of nurse- and/or period-specificity. In this way, an ideal and well-balanced mix of employees is determined not only in terms of skills and employment but also in terms of roster preferences.

3. Solution Procedure

The formulation of this integrated problem based on the original assignment variables (i.e., a nurse is assigned to a certain shift, on a particular day, and to a department) has been decomposed based on the Dantzig-Wolfe decomposition. The problem is solved using an exact branch-and-price approach which simultaneously assigns the nurses to a particular department or to the float unit and constructs a roster over the planning horizon for each nurse. Branch-and-price relies on column generation to find the linear programming relaxation of the master problem. In order to check the optimality of an LP solution, a subproblem, called the pricing problem, is solved to try to identify columns (i.e., individual roster lines) to enter the basis. If such columns are found the LP is re-optimized. Branching occurs when no columns price out to enter the basis and the LP solution does not satisfy the integrality conditions ([2], [12]). Branching is done on the original assignment variables and not on the column variables.

The master problem formulation aims to optimize not only the hospital's objectives (i.e., the quality of care) but also the nurses' objectives. Moreover, the master problem ensures each nurse receives a roster and indicates the best option in case of personnel shortages (e.g., extra personnel, interim personnel, and overtime hours). A small modification in the formulation of the master problem, i.e., each nurse is assigned to at most one schedule, allows us to address nurse recruitment problems using the proposed approach.

The pricing step consists of generating an individual roster line for each nurse for each department (i.e., the involved ward and the float unit) incorporating the dual prices of the coverage requirements of the corresponding department. The pricing problem is a resource constrained shortest path problem which is typically solved using a dynamic programming approach ([6]). This shift assignment needs to be in conformity with all (hard) case-specific time related constraints (e.g., roster rules imposed by law) satisfying the nurses' preferences and contract stipulations and the (soft) regulations as much as possible. Moreover, since the scheduling practices and policies are typically different from ward to ward, this pricing problem is different for each unit. Moreover, this subproblem will even differ from employee to employee since the specific nurse preferences, contract stipulations, work regulations, skill competencies, etc vary largely among the nursing staff. When constructing an individual nurse schedule for a specific ward (and not the float unit), nurses cannot be assigned to shifts of another ward. This practice is only exercised in case of unanticipated variation (e.g., sickness) and is dealt with in the short-term adjustment phase ([1]).

4. Computational results

We test and validate our algorithm on a real-life situation in a Belgian university hospital. Three wards and one float unit are involved in the study. Based on a questionnaire, we reveal the characteristics of the nursing personnel (e.g., preferences, competencies) and the practices and policies of the head nurse. Based on this information gathering, different theoretical and practical insights are obtained on how to increase quality of care and preference satisfaction in case of recruitment of new personnel, re-organization of the existing structure and/or revision of the staffing process of the concerned wards.

References

[1] W.J. Abernathy, N. Baloff, and J.C. Hershey (1971), The Nurse Staffing Problem: Issues and Prospects, *Sloan Management Review* **12**, 87–99.

[2] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W. Savelsbergh, and P.H. Vance (1998), Branch-and-Price: Column Generation for solving huge Integer Programs, *Operations Research* **46**, 316–329.

[3] M.J. Brusco, and M.J. Showalter (1993), Constrained Nurse Staffing Analysis, *Omega* **21**, 175–186.

[4] E.K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem (2004), The state of the art of nurse rostering, *Journal of Scheduling* **7**, 441– 499.

[5] E.K. Burke, P. De Causmaecker, and G. Vanden Berghe (2004), Novel Meta-heuristic Approaches to Nurse Rostering Problems in Belgian Hospitals, In: Leung, J., (ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, CRC Press, 2004, 44.1–44.18.

[6] A. Caprara, M. Monaci, and P. Toth (2003), Models and algorithms for a staff scheduling problem, *Mathematical Programming* **98,** 445–476.

[7] D. Cline, C. Reilly, and J.F. Moore (2003), What's behind RN turnover?, *Nursing Management* **34**, 50–53.

[8] J. Dzubia-Ellis (2006), Float Pools and Resource Teams: A Review of Literature, *Journal of Nursing Care Quality* **21**, 352–359.

[9] F.F. Easton, F. F., D. F. Rossin, and W. S. Borders (1992), Analysis of alternative scheduling policies for hospital nurses, *Production and Operations Management* **1**, 159–174.

[10] A.T. Ernst, H. Jiang, M. Krishamoorty, B. Owens, and D. Sier (2004), Staff scheduling and rostering: A review of applications, methods and models, *European Journal of Operational Research* **153**, 3–27.

[11] W.M., Mercer, (2000), *Attracting and Retaining Registered Nurses – Survey Results*, Chicago IL, 1–8.

[12] F. Vanderbeck (2000), On Dantzig-Wolfe Decomposition in Integer Programming and Ways to Perform Branching in a Branch-and-Price Algorithm, *Operations Research* **48**, 111–128.

[13] R. Venkataraman and M.J. Brusco (1996), An Integrated Analysis of Nurse Staffing and Scheduling Policies, *Omega* **24**, 57–71.

[14] H.W. Warner (1976), Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Approach, *Operations Research* **24**, 842–856.

[15] P.D. Wright, K.M. Bretthauer, and M.J. Côté (2006), Reexamining the Nurse Scheduling Problem: Staffing Ratios and Nursing Shortages, *Decision Sciences* **37**, 39–70.

[16] P. Zurn, C. Dolea, and B. Stilwell (2005), Nurse retention and recruitment: developing a motivated workforce, *The global nursing review initiative (WHO)* **4**, 1–31.

Interactive Evolutionary Multicriteria Scheduling

Jon Marquis, John W. Fowler, Esma Gel
Arizona State University, PO Box 875906 Tempe, AZ 85287-5906, USA,
{Jon.Marquis, John.Fowler, Esma.Gel}@asu.edu

Murat Köksalan
Middle East Technical University, 06531, Ankara Turkey, Koksalan@metu.edu.tr

Pekka Korhonen, Jyrki Wallenius
Helsinki School of Economics, Runeberginkatu 22-24 00100 Helsinki, Finland,
{Pekka.Korhonen, Jyrki.Wallenius}@hse.fi

1 Introduction

Scheduling problems exist in almost all manufacturing industries since the job processing order significantly impacts both the amount of work in process and the on-time delivery of products. Many real-world scheduling problems, however, are not represented by a single criteria. Instead, decision makers must trade off conflicting objectives to create an acceptable schedule.

Unfortunately, making these trade-offs raises two issues. First, many scheduling problems are NP-hard even for a single criterion. Adding additional criteria only makes the problems harder (and more time consuming) to solve to optimality. Second, the way a decision maker (DM) combines conflicting criteria is often difficult to capture. In some cases, the decision maker cannot quantify how the objectives combine into a preference function that can then be optimized. The typical approach is to generate the set of Pareto Optimal solutions (or approximate Pareto Optimal solutions) and present them to the DM. The DM then selects a solution *a posteriori*.

Our approach addresses both of these issues simultaneously by applying an interactive genetic algorithm which builds upon the work of [5]. They demonstrate an interactive genetic algorithm that supports linear combinations of the objective functions. We use a similar interactive genetic algorithm framework, but our technique will support any quasi-concave function of the objectives. Incorporating the DM preferences in the metaheuristic via DM interaction guides the search to the most preferred region of the solution space, unlike approaches which attempt to generate the entire set of Pareto Optimal solutions.

We then apply this framework to a bicriteria scheduling problem which would generally require a metaheuristic to generate a good solution in a reasonable amount of time. This approach combines the proven ability of genetic algorithms to find nearly optimal solutions to complex problems with the restricted search of the solution space provided by interactive decision making techniques. The remaining sections of this abstract describe the scheduling problem we are solving and the structure of the genetic algorithm.

2 Demonstration Problem: $Pm||\Sigma w_j C_j$, ΣT_j

To demonstrate the effectiveness of this approach, we apply our technique to a problem considering both the sum of the weighted completion times and the total tardiness in an identical parallel machine environment. Minimizing weighted completion times is equivalent to minimizing the mean weighted flow times when all jobs are ready at time zero. Minimizing total tardiness maximizes on-time delivery. Since both of these criteria are NP-hard and the optimal solution to one objective is often far from optimal for the other, understanding the trade-off between the objectives is critical.

Our approach is to use an interactive genetic algorithm (GA) to assign jobs to machines. Once jobs are assigned to machines, a standard single machine scheduling method (*e.g.*, ATC - Apparent

Tardiness Cost [6]) is used to order the jobs on each machine. The following section details the steps of the interactive algorithm and explains how the user's preference information is captured.

3 Interactive Genetic Algorithm

The genetic algorithm we use is modified from programs such as GAlib [7] and general techniques like those presented in [4] in two ways. Rather than using a fitness function, we put the population in preference order by using convex preference cones [3] to compare the solutions. These preference cones are formed by asking the decision maker to find the best solution from a small subset of the population.

The preference cones are used to compare population members (solutions) in the objective space. A solution can have four possible locations relative to a cone, illustrated by the four numbered points in Figure 1. The white solutions define the cone and the black solutions are being compared to the cone. The lemmas in [3] explain the relationships used to find a solution's location relative to a cone.

The interactions with the decision maker guide the algorithm according to the DM's preferences, allowing us to explore preferred solutions rather than attempt to generate the full (near) Pareto-optimal set. Using convex preference cones allows us to support any quasi-concave preference function of the scheduling criteria without asking the decision maker to articulate the preference function. The steps of the algorithm are described below.

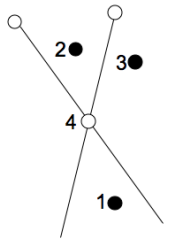


Figure 1: Possible locations of a population member with respect to a cone

3.1 Create the Population

The initial population is widely recognized as being important to limiting the cognitive and computing load required to arrive at an acceptable solution using an interactive genetic algorithm [5]. In this case, a good option is to assign jobs to machines with equal probability.

3.2 Interact with the Decision Maker

The interactions with the decision maker consist of asking for the best of a subset of the population. The DM is called periodically during the course of the algorithm, and finding a reasonable interaction budget and the best way to allocate that budget is an ongoing research topic.

3.3 Preference Order the Population Members

Ordering the population occurs in two steps. First, the population is clustered by rank [2]. In this clustering, all Pareto-nondominated members are placed before the Pareto-dominated members of the population. The second step uses the preference cones to order the solutions in the nondominated cluster. (Ordering only the most preferred cluster reduces the algorithm's computational load.) The first solution in the ordered population is reported when the stopping criteria is met.

3.4 Perform Crossover Operation and Mutate the Children

The first step in a crossover operation is to select parents. The first parent is selected probabilistically, with higher selection probabilities going to more preferred solutions. The second parent is selected randomly from the population. The parents are bred using a one point crossover operator [4] applied at a random location to combine the two parents, as in [1]. Once the crossover operation is complete, the children are mutated with probability P. We employ a mutation operator that randomly interchanges the machine assignments of two jobs in the child.

3.5 Update the Population

The children produced in the crossover and mutation process replace the worst 20% of the population. Once the new members are in place, the population is examined to find the best solution with knowledge of the true preference function. This information is retained for testing purposes only and is never used in making decisions within the algorithm. Unless the stopping criteria has been met, the GA returns to step 2 or 3 (depending on whether or not another DM interaction is called for.)

3.6 Stopping Criteria

The algorithm currently stops once a preset number of generations have been created. We will also consider a modified convergence stopping criterion based on the algorithm reaching satisfactory stable solution.

4 Computational Results

We tested the effectiveness of this algorithm by randomly generating scheduling problem instances and objective function coefficients. Each job is assigned a processing time, weight, and due date. The jobs are randomly assigned to machines and ordered using the ATC heuristic [6]. The values of total tardiness and weighted completion time are calculated for each solution. Our goal is to minimize the user's preference function, which is assumed to be a quasi-concave function of the objectives.

To evaluate our results, the algorithm keeps track of the best solution found with knowledge of the true preference function. The preference function value of the output solution is compared to this value to provide us with an estimate of how well the convex cones sort the population. Specifically, we compare the difference between the best found solution and the output solution as a percentage of the best found solution. This percentage provides a reasonably problem instance independent evaluation of the proposed technique. In all cases where the DM was consulted the result was zero, meaning the algorithm found the output solution was equal to the best solution the GA found.

Since an interactive GA can be thought of as a guided search, we also report the average value of the best solution. As this average decreases with increased interaction with the DM, it is clear that the guided search approach works well for these problems and allows us to report a good solution without generating the complete set of Pareto Optimal solutions. These results are shown in figure 2 below.

				Average Output Solution		
Machines	Jobs	Generations	Preference Function	0 DM Calls	3 DM Calls	6 DM Calls
3	200	240	Linear	52646.6	20.2	20.2
3	100	120	Linear	16680.9	22.4	22.4
3	50	60	Linear	6832.5	28.6	29.9
4	200	240	Linear	96970.5	61787.9	20.2
4	100	120	Linear	22768.3	10466.6	22.4
4	50	60	Linear	10591.5	4918.5	28.6
3	200	240	Chebyshev	49556.7	20.2	20.0
3	100	120	Chebyshev	16158.7	22.4	22.4
3	50	60	Chebyshev	6650.0	28.6	29.3
4	200	240	Chebyshev	90154.5	58280.0	26676.9
4	100	120	Chebyshev	21405.3	9921.9	22.4
4	50	60	Chebyshev	10134.2	4777.6	28.6

Figure 2: Average value of the output solution (smaller is better)

5 Conclusions

We have described an interactive multiobjective GA that uses convex cones to evaluate solutions without knowledge of the user's true preference function. This algorithm is applied to a bicriteria scheduling problem which demonstrates that the convex cones are capable of sorting the population, and that the DM interactions guide the algorithm into more preferred regions of the solution space, improving the solution without spending additional computational effort.

References

[1] H. Balasubramanian, L. Monch, J. Fowler and M. Pfund (2004), Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness, *International Journal of Production Research*, **42**(8), 1621 – 1638.

[2] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan (2002), A fast and elitist multiobjective genetic algorithm: Nsga-ii, *IEEE Transactions on Evolutionary Computation* **6**(2), 182 – 197.

[3] P. Korhonen, J. Wallenius and S. Zionts (1984), Solving the discrete multiple criteria problem using convex cones, *Management Science* **30**(11), 1336 – 1345.

[4] Z. Michalewicz (1996), *Genetic Algorithms + Data Structures = Evolution Algorithms, 3rd Edition*, Springer.

[5] S.P. Phelps and M. Köksalan (2003), An interactive evolutionary metaheuristic for multiobjective combinatorial optimization, *Management Science* **49**(12), 1726 – 1738.

[6] A. Vepsalainen and T. Morton (1987), Priority rules for job shops with weighted tardiness costs, *Management Science*, **33**(8), 1035 – 1047.

[7] M. Wall (1995), Massachusetts Institute of Technology's GAlib Library, http://web.mit.edu/galib/www/GAlib.html.

Constructive versus Improvement Heuristics: An Investigation of Examination Timetabling

Edmund K. Burke, Graham Kendall
University of Nottingham, United Kingdom
Barry McCollum, Paul McMullan
Queen's University of Belfast, United Kingdom, p.p.mcmullan@qub.ac.uk

1. Introduction

Examination timetabling has been a widely studied research problem for over 30 years, with many techniques being used to produce good quality solutions. The majority of the techniques rely on producing an initial solution using a constructive heuristic, and then improving it via a local search [1,2,3], relying on the correct setting of internal parameters to allow generally acceptable results for different data sets. Unfortunately, in attempting to provide good quality results for all data sets, in many cases we may not achieve the best possible solutions. From the author's practical experience, it has been observed that human intervention is often necessary to adapt the scheduling techniques used for the particular characteristics of each data set. The goal of this paper is to remove this human intervention where possible, and allow full automation for all problem instances. We investigate a number of benchmark problem data sets, of varying degrees of complexity, to see if there is a relationship between the difficulty of the problem and the effects of combining constructive and improvement heuristics in achieving the best possible solution under specified time constraints. In this way we can attempt to generalise the link between the problem instances and the approach taken in gaining a solution. We use the Carter benchmark datasets [4], which represent real world instances and whose characteristics in terms of measure of difficulty are well known. We use the commercial examination timetabling system, Optime [5] to generate the results. Our results show that there is a relationship between the difficulty of the problem and the time spent in each part of the solution methodology.

2. Data Sets

We are using the Carter benchmark data sets [4] as these are widely used in other work. The difficulty measure is based on the *conflict density* value, calculated as the average number of all other exams that each exam conflicts with, divided by the total number of exams. For instance, a density of 0.25 denotes that, on average, each exam conflicts with 25% of the other exams. It is proposed that this is an obvious initial starting point in establishing a correlation between difficulty and appropriately tailored scheduling heuristics. If a link can be established this, could lead to a more fine-grained analysis of the characteristics of data sets in relation to various parameter settings for existing scheduling techniques, and can build on current research into the categorisation and similarity measure analysis of examination data sets [6].

3. Solution Methodology

In general, successful scheduling techniques have employed a two-phase approach to establishing a timetable; construction to produce a feasible solution and improvement, employing intelligent search techniques to find high quality solutions given specified objectives. It has been observed that if we continue construction beyond the point at which we have produced a feasible solution that we often achieve a better quality solution on which the improvement phase can then operate [7]. In conjunction with an improvement phase, an even higher quality solution could be possible. However, in practice the same relative time spent on construction and improvement respectively may not yield the best results (given limiting factors such as time constraints) for each data set. For

instance, one data set may favour more construction over improvement, while another will favour a greater emphasis on improvement. At this point human intervention or trial and error are required, and we cannot say the process is truly automatic.

The main purpose of this study is to ascertain if the relative time spent in each of the two phases relates to the difficulty of the problem instance, with the aim of generalising the process without the trade-off of solution quality. The data sets used have a wide range of conflict density values, which will help to identify clear distinctions between difficulty and results based on the tested combinations of construction / improvement heuristic under analysis.

The Optime examination system uses this two phase approach when generating timetable solutions for examination data sets. Initial solution construction is carried out by an Adaptive (Squeaky-Wheel) ordering heuristic [7] technique. This utilises a weighted order list of the exams to be scheduled, the initial ordering based on the *degree* (number of conflicts) of each exam. This allows an initial estimation of a list with the most difficult exams being placed first. Each exam weighting is then increased, depending on the difficulty or penalty of its placement in the schedule, which allows the ordering to adapt as difficult exams are encountered. Although construction techniques in general are simply used to establish a feasible solution, the adaptive heuristic can continue to improve a feasible solution as the exam ordering changes due to the weighting.

The improvement phase takes the feasible timetable from the constructive heuristic and implements the Great Deluge Algorithm [8] as a local search method. The Great Deluge (also known as Degraded Ceiling) was introduced by Dueck [9] as an alternative to Simulated Annealing [10,11]. This uses a boundary condition to accept worse solutions, in order to escape local optima. The boundary is initially set slightly higher than the initial solution cost, and gradually reduced throughout the improvement process. New solutions are only accepted if they improve on the current cost evaluation or are within the boundary value. This approach has previously been successfully applied to construction and improvement techniques in timetabling problems [8].

The evaluation function to drive both construction and improvement in the search for improved solutions is that used within the commercial examination timetabling system, Optime [5]. It is used in preference to that traditionally used for the Carter data sets [4], as the experiments were run through the Optime software. This allows many more constraints than those included within the Carter evaluation function to be included, in order to extend the analysis to further data sets. The Carter evaluation function is primarily concerned with minimization of the proximity penalties caused by exams which are scheduled a specified number of periods apart. Each occurrence of a student taking exams within proximity limits will add to the total penalty. The Optime evaluation function [5] takes this proximity measure into account, as well as accommodating many other soft constraints considered in real-world scheduling problems.

4. Results

Initial results are encouraging in being able to establish a link between construction and improvement. The experiments have involved allocating a given number of total solution generations for the entire process, and dividing this allocation by stepped percentages for construction and improvement respectively. For example, one test run for a single data set would involve an initial setting of 0% of the total evaluations spent on construction, 100% spent on improvement, then 10% on construction and 90% on improvement and so on. This is repeated for a single data set to get an average or typical set of results, and performed for each of the data sets under consideration.

Figure 1 presents the initial results achieved from the average of three test runs for each data set. The construction value is presented as a value between 0 and 10, representing 0% to 100% of time spent on construction within the entire test run. The construction value achieved is that setting at which the best quality solution was found for each of the 11 different combinations of construction and improvement time. As can be seen, there is a direct (inversely proportional) correlation between construction time to the difficulty (in terms of conflict density) of the data set.

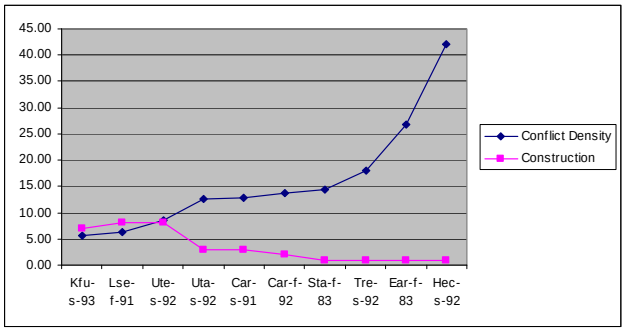


Figure 1 – Data Sets used in this Study

5. Discussion and Conclusions

We have shown there is a link between the difficulty (in terms of conflict density) of examination data sets and the relative time spent on construction and improvement when creating timetable solutions. Each different scheduling technique has many parameter settings, with a certain dependence on the human user to provide the right balance. The work outlined provides scope for further analysis in removing this dependency. Further work will be to investigate whether this observation holds for actual real-world examination data sets, drawn from our commercial scheduling package.

References

[1] E.K. Burke, M. Trick (eds), (2005), The Practice and Theory of Automated Timetabling V, *Revised Selected Papers from the 5th International conference, Pittsburgh 2004*, Springer Lecture Notes in Computer Science **3616**, Springer 2005.

[2] G.M. White, B.S. Xie, S. Zonjic (2004), Using tabu search with longer-term memory and relaxation to create examination timetables, *European Journal of Operational Research*, **153**(16), 80 – 91.

[3] R. Qu, E.J. Burke, B. McCollum, L. Merlot, Lee S. (2006), A Survey of Search Methodologies and Automated Approaches for Examination Timetabling, *Computer Science Technical Report* No. NOTTCS-TR-2006-4.

[4] M.W. Carter, G. Laporte, S. Lee, (1996), Examination timetabling: Algorithmic strategies and applications, *Journal of the Operational Research Society* **47**(3), 373 – 383.

[5] E.K. Burke, G. Kendall, B. McCollum, P.J.P. McMullan, J.P. Newall, (2006), A Preference Based Measurement of Optimisation, *ASAP Technical Report* eMAP/2006/02a.

[6] Y. Yang, S. Petrovic, (2004), A Novel similarity measure for heuristic scheduling in examination timetabling, *Selected Papers from 5th International conference on the Practice and Theory of Automated Timetabling*, Springer Lecture Notes in Computer Science **3616**, 377 – 396.

[7] E.K. Burke, J.P. Newall (2004), Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings, *Annals of Operational Research* **129**, 107 – 134.

[8] E.K. Burke, Y. Bykov, J.P. Newall, S. Petrovic (2003) A Time-Predefined Approach to Course Timetabling, *Yugoslav Journal of Operational Research (YUJOR)* **13**(2), 139 – 151.

[9] G. Dueck (1990), Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing, *Journal of Computational Physics* **90**, 161 – 175.

[10] S. Kirkpatrick, J.C.D. Gellat, M.P. Vecci (1983), Optimization by Simulated Annealing, *Science* **220**, 671 – 680.

[11] E. Aarts, J. Korst, W. Michiels (2005), Simulated Annealing, In E.K. Burke, G. Kendall (eds), (2005) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies*, Springer 2005, Chapter 7, 187 – 210.

Real-time Student Sectioning

Keith Murray, Tomáš Müller
Purdue University, West Lafayette IN 47907, USA, {kmurray, muller@purdue.edu}

1 Introduction

Student sectioning is often considered as a subproblem in course timetabling. Once a timetable has been developed, the object is to assign students to specific sections of courses in order to minimize conflicts. Several approaches have been applied to this problem [1, 3], often iterating between sectioning and timetabling during the solution process. There have also been experiments with more interactive approaches such as course bidding systems [4]. One thing all of these techniques have in common is that the optimization is performed on all student schedules at a single point in time. There is often a need, however, to section additional students to classes or make schedule revisions. In the system being developed at Purdue University [2], the timetable is created and most students are sectioned based on demand data in student schedule requests, but the course requests of beginning students are unknown at the point in time when the timetable is created.

2 Sectioning during the Timetabling Process

The real-time sectioning algorithm introduced here has been designed to be used in conjunction with an optimized batch sectioning conducted at the end of the course timetabling process. Construction of the timetable considers actual course demand from students who have submitted schedule requests plus the projected demand for students who are anticipated to enroll. This projected demand is in the form of a conflict matrix describing which courses must be taken in common for each student. At the conclusion of the timetable construction it is therefore possible to identify class assignments made to actual students versus class space requirements generated on the basis of enrollment projections. For every class, this results in space being allocated to: (1) scheduled students and (2) future students. Some class space may also remain unallocated (see Fig. 1).

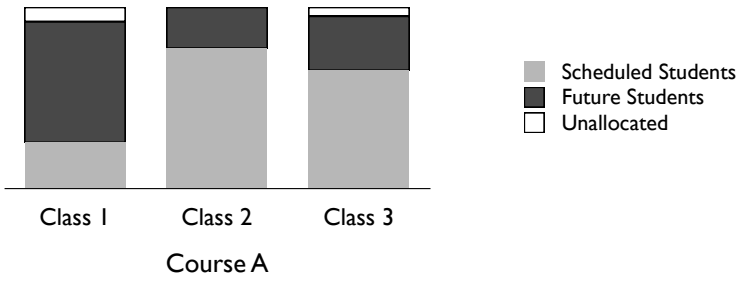


Figure 1: Illustration of possible space allocation in three equal sized class sections associated with a course at conclusion of timetabling process.

The extent to which individual classes are filled, or have space available, depends upon the combinations of courses selected by students, their time preferences, and conflicts caused by how individual classes overlap in time. There should be fewer student conflicts with an efficient timetable.

3 Real-time Sectioning

The real-time sectioning algorithm is used with the class spaces allocated to future students plus any unallocated spaces. Class space is essentially reserved for students who are anticipated to submit scheduling requests at a later date (e.g., first year students, transfers, procrastinators). Unallocated spaces are available either to these projected future students or for currently scheduled students who wish to make class changes.

As students submit schedule requests, each course is ranked in priority order. During real-time sectioning, the search for individual student class schedules employs a backtracking process considering possible assignments beginning with those classes associated with the student's highest priority course. As it evaluates each possible assignment, it compares class spaces available (i.e., those for future students or unallocated) with a projection of the demand for each class by later enrolling students. This difference between available class spaces in the timetable and the expected need for each class time is used to direct students away from class assignments that would result in excess demand.

The expected demand for each class is calculated by examining the combinations of courses taken by students enrolled during the previous like term. For example, if the pattern of courses taken by a student in the previous term would require the student to be sectioned into a class at one specific time in the current timetable, then one student is added to the expected need for that class. If the student could feasibly have been assigned to either of two different classes associated with a course offering, then one half student is added to the expected demand for each of the two classes. Furthermore, if the projected number of enrollments within certain categories of students is anticipated to be greater or less than in the previous term, the expected demand generated by each student may be weighted by the ratio of projected to past students in that category. In this way the expected need for each class can be pre-computed before the real-time sectioning process begins.

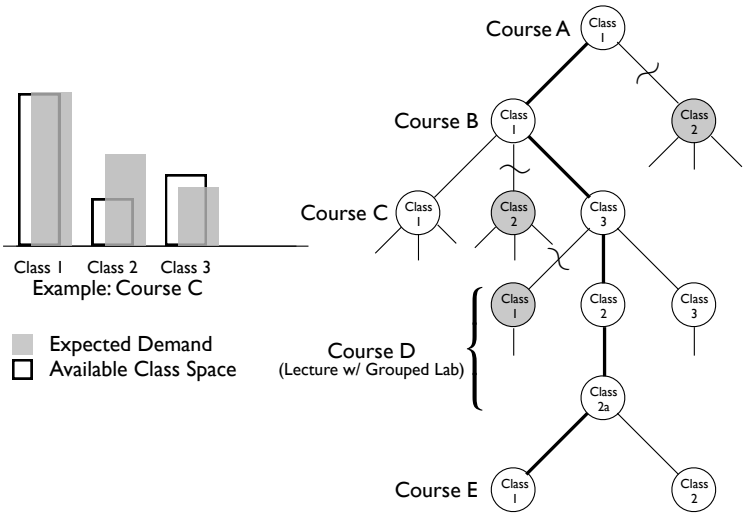


Figure 2: Illustration of use of expected demand in search process. Search progresses in order of priority assigned to each course (A to E). For Course C, Class 2 has expected demand greater than available space so it is only included if no other option exists. Course D illustrates case where a required grouping between two parts of a course restricts options.

The first step for the sectioning algorithm is to determine which class spaces are available to a student. This is based on student attributes (e.g., program area, semester classification) and any course or class specific reservations. The courses a student is eligible to attend are sorted in priority order and a branch and bound search is then conducted for the schedule that best satisfies the student's request (see Fig. 2). From among the available classes, the object is to assign the student to a set that fulfills the student course request and is least likely to create time conflicts for students who will be scheduled later. Other student preferences, such as the need for free time, or for a compact versus a distributed schedule, can also be considered in the search.

Potential class assignments within a course where the expected demand is greater then the available space are assigned a penalty based on the amount of excess demand. Branches beginning with such classes only need to be searched if the total penalty for the best current solution is greater than the penalty for including this class. In no case is an eligible student blocked from scheduling a course offering as a result of expected future demand. As students are assigned to specific classes during the sectioning process, the expected demand for each class is adjusted to reflect the assignment. The availability of space in each class and the expected demand are thus dynamically adjusted throughout the process.

4 Conclusions

An algorithm for allowing real-time sectioning of students to a class timetable has been proposed that allows spaces to be maintained in classes at appropriate times to meet the expected demand by all students. This process may be summarized as: (1) initial availability of class spaces and potential demand at each time is determined as a result of automated timetabling process based on actual and expected student demand; and (2) class assignments for additional students are optimized based on the potential an assignment will create future conflicts, determined by the difference between space available and expected need, and by student course and time preferences. The result is a sectioning process that can take advantage of what is known about the combinations of courses taken by students to avoid making assignments that prevent later students from enrolling in classes they may need. While it is impossible to achieve a truly optimal sectioning using this approach, it is believed that it will significantly improve on the section balancing method currently utilized. This system is currently in the process of development with the expectation that experimental results using actual timetables and student course requests will be available for presentation and discussion in the near future.

References

[1] M.W. Carter and G. Laporte (1998), Resent developments in practical course timetabling, In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, 3 – 19. Springer-Verlag LNCS 1408.

[2] K. Murray, T. Müller, and H. Rudová (2007), Modeling and solution of a complex university course timetabling problem, In Edmund Burke and Hana Rudová, editors, *Practice and Theory of Automated Timetabling VI*, Springer-Verlag, In press.

[3] A. Schaerf (1999), A survey of automated timetabling. *Articifial Intelligence Review* **13**(2), 87 – 127.

[4] Tayfun Sönmez and M. Utku Ünver. Course bidding at business schools. Technical report, Koc University, 2004.

Staff Rostering Problem: Choice of a Mathematical Model

Édith Naudin, Peter Chan, Michael Hiroux, Georges Weil, Tahar Zemmouri
Equitime, 1 allée de Certèze, 38610 Gières, France, {enaudin, pchan, mhiroux, tzemmouri, gweil}@equitime.fr

1 Introduction

Since the first use of the Dantzig-Wolfe decomposition ([2]) by Gilmore and Gomory ([5]) on the cutting stock problem, models based on column generation became more and more popular.
In this context, we propose to compare the computational behavior of several mathematical models for the same problem: the staff rostering problem. The first model, considered to be the most conventional, uses decision variables containing the least information. The other models are obtained through a Dantzig-Wolfe decomposition of the first. The variables being more complex, these models often require column generation to be solved (see [1], [4] and [6]).
The staff rostering problem solved here consists of assigning workers to tasks whose starting and ending times are fixed, and subject to constraints defined over different horizons. The first model uses assignment variables; the other models use variables representing rosters with different horizons.
Let I be the set of workers to be assigned over the set of tasks T, defined over a set of consecutive days $D = \{1, \ldots, |D|\}$. Each task $t \in T$ is characterised by:

- a day $day_t \in D$
- a starting time $b_t \in \mathbb{N}$
- an ending time $e_t \in \mathbb{N}$
- a duration $d_t = e_t - b_t \in \mathbb{N}^*$
- a staffing requirement $r_t \in \mathbb{N}^*$
- a positive cost in case of understaffing: $c_t \geq 0$ per person missing

A solution to this problem must respect the following constraints: The staffing requirement of each task is to be covered as completly as possible (1)
The set of assignments of each worker must verify:

- At all instants, each worker is assigned to at most one task; (2)
- Two consecutive working days are separated by a rest period of which the minimal duration is $DayR$. (3)
- The working time per day is limited to a maximum of $DayW$; (4)
- The total working time is limited to a maximum of $WeekW$; (5)

As we can see, the problem has constraints expressed over different horizons:

- Short-horizon constraints applied over a task: (1);
- Medium-horizon constraints applied over one day: (2) and (4);
- Long-horizon constraints applied over several days being scheduled: (3) and (5).

2 The Models

2.1 First Model: Short-term variables

The first model uses the following boolean assignment variables:
$\forall i \in I, \forall t \in T, x_{i,t} \in \{0;1\}, x_{i,t} = 1$ if the person i is assigned to the task t, otherwise, 0 .
The following variables are used:
$\forall t \in T, s_t \in \mathbb{N}$ is the understaffing of the task t. We define the following sets:

- $INC_0(T) \subseteq T \times T$ the list of pairs of incompatible tasks for the constraint (2):
  $INC_0(T) = \{(t_1, t_2), t_1, t_2 \in T, day_{t_1} = day_{t_2}, b_{t_1} \leq b_{t_2} \text{ and } b_{t_2} < e_{t_1}\}$
- $INC_1(T) \subseteq T \times T$ the list of pairs of incompatible tasks for the constraint (3):
  $INC_1(T) = \{(t_1, t_2), t_1, t_2 \in T, day_{t_1} + 1 = day_{t_2}, b_{t_2} < e_{t_1} + DayR\}$
- $\forall d \in D, LD_d \subseteq T$ the list of tasks on each day d:
  $\forall d \in D, LD_d = \{t \in T, day_t = d\}$

$$
(P) \begin{cases}
\min & \sum_{t \in T} c_t.s_t & & \\
\text{s.t.} & & & \\
& \sum_{i \in I} x_{i,t} + s_t \geq r_t, & \forall t \in T & (1) \\
& x_{i,t_1} + x_{i,t_2} \leq 1, & \forall i \in I, \forall (t_1, t_2) \in INC_0(T) & (2) \\
& x_{i,t_1} + x_{i,t_2} \leq 1, & \forall i \in I, \forall (t_1, t_2) \in INC_1(T) & (3) \\
& \sum_{t \in LD_d} d_t.x_{i,t} \leq DayW, & \forall i \in I, \forall d \in D & (4) \\
& \sum_{t \in T} d_t.x_{i,t} \leq WeekW, & \forall i \in I & (5) \\
& x_{i,t} \in \{0;1\}, & \forall i \in I, \forall t \in T & \\
& s_t \in \mathbb{N}, & \forall t \in T &
\end{cases}
$$

The set of constraints (1) computes the understaffing s_t of tasks. Each task t induces a linear cost c_t when it is understaffed. (P) computes a planning which satisfy the sets of contraints (2) - (5) and which minimize the weighted understaffing of tasks.

2.2 Other Models

2.2.1 Second model: mid-term variables

The second model can be obtained by applying a Dantzig-Wolfe decomposition on the first model. It uses assignment variables that represent valid daily rosters. The resource constraints over workers (2) and the limited daily working hours constraints (4) are respected implicitly by the variables. The other problem constraints are handled as linear constraints in the master problem (MP).
Given the huge number of possible valid daily rosters, the resolution of this model relies on the column generation technique. For each day $d \in D$, the sub problem is a constrained shortest path problem in a graph G_d where the nodes represent the tasks of the day and the arcs link compatible tasks: $G_d = (N_d, A_d)$, where $N_d = LD_d\}$ and
$A_d = \overline{INC_0(LD_d)} = \{(t_1, t_2), t_1, t_2 \in LD_d, e_{t_1} \leq b_{t_2}\}$.
The resource constraint (2) is expressed by the graph construction . The daily maximum work time constraint (4) is processed like a constraint of capacity in the constrained shortest path problem and considered during the construction of rosters. To find rosters satisfying the constraints (4) in

the graphs G_d, a dynamic programming algorithm is used.This form of decomposition is interesting when the constraints which are transferred in the variables are so difficult to put into a linear form in the first model that one can gain in the quality of the lower bound obtained from a linear relaxation. For instance, the constraint (2), expressed with the inequality $x_{i,t_1} + x_{i,t_2} \leq 1$ in (P) is a linear form of the following quadratic constraint: $x_{i,t_1}.x_{i,t_2} = 0$. The quadratic form gives a better lower bound, but is not easy to use: with the linear form, we can use the simplex algorithm to compute the lower bound. The lost of quality with the linear form can be recovered by using the second model: each roster verify the constraint (2). Unfortunately, this model has a cost: the sub-problem (SP) for column generation is NP-hard (Dror, [3]).

2.2.2 Third model: long term variables

The third model - also obtained by a Dantzig-Wolfe decomposition - uses variables that represent valid rosters over the whole horizon. The master problem is a set covering problem with the staffing requirements (1). The other constraints are expressed by the validity of rosters. Here the sub-problem for column generation is a constrained shortest path problem in a graph G where the nodes represent the tasks over the whole horizon and the arcs join compatible tasks:
$G = (N, A)$, where $N = T$ and $A = \overline{INC_0(T) \cup INC_1(T)} = \{(t_1, t_2), t_1, t_2 \in T, e_{t_1} \leq b_{t_2}\} \setminus \{(t_1, t_2), t_1, t_2 \in T, day_{t_1} + 1 = day_{t_2}, b_{t_2} < e_{t_1} + DayR\}$. The resource constraints (2) and the minimum daily rest duration constraint (3) are satisfied by construction of the graph.
The other constraints (maximum daily working time (4) and over the whole horizon (5)) are handled as constraints of capacity and considered during the column generation.
The following table resumes the processing of each constraint in the three models:

Contrainte		Model 1	Model 2	Model 3
Staffing requirements	(1)	Master Problem	Master Problem	Master Problem
Resource Constraint	(2)	Master Problem	SP: Graph	SP: Graph
Daily rest duration	(3)	Master Problem	Master Problem	SP: Graph
Daily work duration	(4)	Master Problem	SP: Capacity	SP: Capacity
Total work duration	(5)	Master Problem	Master Problem	SP: Capacity

In our presentation, we will compare the computational behavior of these models in different scheduling scenarios.

References

[1] C. Barnhart, E. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance (1998), Branch-and-Price: Column Generation for Solving Huge Integer Programs, Operations Research **46**, 316 – 329.

[2] G.B. Dantzig and P. Wolfe (1960), Decomposition Principle for Linear Programs, *Operations Research* **8**, 101 – 111.

[3] M. Dror, Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW, *Operations Research* **42**, Technical Note, 977 – 978.

[4] M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers (1999), A Column Generation Approach for Large Scale Aircrew Rostering Problems, *Operations Research* **47**, 247 – 263.

[5] P.C. Gilmore and R.E. Gomory (1960), A linear programming approach to the cutting stock problem, *Operations Research* **9**, 849 – 859.

[6] B. Jaumard, F. Semet, and T. Vovor (1998), A generalized linear programming model for nurse scheduling, *European Journal of Operational Research* **107**(1), 1 – 18.

Online Scheduling of Parallel Jobs on Two Machines is 2-Competitive

Jacob Jan Paulus, Johann L. Hurink
University of Twente, P.O. box 217, 7500AE Enschede, The Netherlands, j.j.paulus@ewi.utwente.nl

We consider online scheduling of parallel jobs on parallel machines. For the case with two machines and the objective of minimizing the makespan, $P2|online-list, m_j|C_{max}$, we show that 2 is a lower bound on the competitive ratio of any online algorithm. Thereby we not only improve the existing lower bound of $1+\sqrt{\frac{2}{3}}$, but also close the gap with the trivial upper bound of 2.

1 Introduction

In recent years the problem of scheduling parallel jobs on parallel machines gained considerable attention. Contrary to classical parallel machine scheduling problems, jobs may require processing on several machines in parallel. Applications, like computer architectures with parallel processors, motivate the study of these type of scheduling problems. For an overview of recent developments on this type of scheduling problems see Johannes [1].

We study the problem of online scheduling of parallel jobs on two parallel machines. Jobs arrive one by one and are characterized by their processing time and the number of machines simultaneously required for processing (1 or 2 machines). As soon as a job arrives, it has to be scheduled irrevocably without knowing the characteristics of the future jobs. Preemption is not allowed and the objective is to minimize the makespan. This problem is denoted by $P2|online-list, m_j|C_{max}$. We show that no online algorithm for this problem can have competitive ratio strictly less than 2. In such a $online-list$ model the online algorithm has to schedule the jobs as they appear in the list, but every job is available from time 0 in the schedule.

The online scheduling of parallel jobs on two machines has previously been studied by Chan et al. [3]. They proved a lower bound of $1+\sqrt{\frac{2}{3}}$ on the competitive ratio of any online algorithm. For the case where jobs arrive in non-decreasing order of processing time, they give an optimal $\frac{3}{2}$-competitive algorithm. And for the case where jobs arrive in non-increasing order of processing, they give a $\frac{4}{3}$-competitive algorithm and a lower bound of $\frac{9}{7}$ on the competitive ratio of any online algorithm. For the general problem, with an arbitrary number of machines, $P|online-list, m_j|C_{max}$, Johannes [1] was the first to develop an online algorithm with constant competitive ratio. She gave a 12-competitive online algorithm, which was later improved by Ye and Zhang [2] to an 8-competitive algorithm.

For the problem with two machines, a greedy algorithm which schedules the jobs upon arrival as early as possible, has a competitive ratio of at most 2. This follows directly from the fact that never both machines are left idle by such a greedy algorithm. To prove that there is no online algorithm with competitive ratio strictly less than 2, we construct a series of job sequences in which jobs have an alternate machine requirement of 1 and 2, and show that no online algorithm can have competitive ratio strictly less than 2 for these sequences. Therefore, the greedy algorithm is the best possible for the considered online problem with two machines.

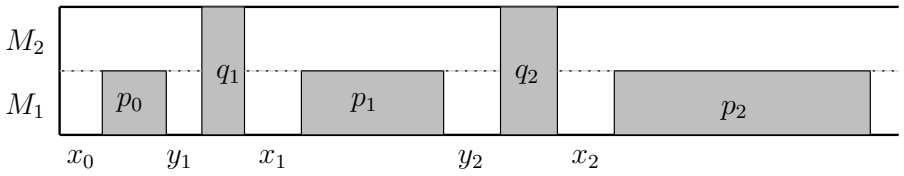


Figure 1: Structure of the online schedule for σ_2

2 Lower Bound on the Competitive Ratio

To prove a lower bound of 2 on the competitive ratio of any online algorithm for $P2|online - list, m_j|C_{max}$, we are going to construct a series of job sequences and argue that no online algorithm can have a makespan strictly less than twice the makespan of the optimal offline solution. We assume ON to be an online algorithm with competitive ratio strictly less than 2, and show that such an algorithm cannot exist. By $C_{OPT}(\sigma)$ and $C_{ON}(\sigma)$ we denote the makespan of the optimal offline schedule and the makespan of the schedule constructed by the online algorithm ON on the job sequence σ, respectively.

We construct a sequence of jobs as an adversary. We observe how the online algorithm schedules the last job to determine the characteristics of the next job. The first job p_0 has unit length and machine requirement 1. Depending on the start time x_0 of this job we define the second job q_1 to have length $x_0 + \epsilon$ and machine requirement 2. Due to its characteristics this second job can only be scheduled after the first (see Figure 1). The next job p_1 will have length $x_0 + p_0 + y_1 + \epsilon$ (where y_1 is the delay between the completion of the first and the start of the second job) and machine requirement 1. This job can only be scheduled after q_1, and so on. The job sequence σ_n the adversary constructs is of the form $(p_0, q_1, p_1, q_2, p_2, \ldots, q_n, p_n)$, where p_i (q_i) denotes a job with processing time p_i (q_i) and a machine requirement of 1 (2). The job lengths are defined as:

$$\begin{aligned} p_0 &= 1 \\ p_1 &= x_0 + p_0 + y_1 + \epsilon \\ p_i &= 2 \cdot p_{i-1} \quad \forall i \geq 2 \\ q_1 &= x_0 + \epsilon \\ q_i &= \max\{y_{i-1}, q_{i-1}, x_{i-1}\} + \epsilon \quad \forall i \geq 2, \end{aligned}$$

where x_i and y_i are values given by delays the online algorithm has used for placing earlier jobs. Therefore, the job lengths are depending on the online algorithm ON.

First we prove that any online algorithm with competitive ratio strictly less than two has to schedule the jobs in the same order as they appear in the sequence σ_n. As a consequence, Figure 1 illustrates the structure of the online schedule. Therefore, the only remaining decision for the online algorithm ON is to decide how long it delays the start of a job, i.e. how much time is left between the start of the current job and the completion of the previous job. We denote by x_i (y_i) the delay incurred by ON on job p_i (q_i), completing thereby also the definition of the processing times q_i.

Using this result, the structure of the online schedule for σ_n is fixed and its makespan is given by:

$$C_{ON}(\sigma_n) = x_0 + p_0 + \sum_{i=1}^{n}(y_i + q_i + x_i + p_i)$$

The optimal schedule for σ_n is obtained by scheduling the jobs $p_0, ..., p_{n-1}$ parallel to job p_n after a block containing the jobs $q_1, ..., q_n$. The makespan of the optimal schedule is, therefore, given by:

$$C_{OPT}(\sigma_n) = \sum_{i=1}^{n} q_i + p_n$$

Using these makespans for the job sequence σ_n, we can calculate the competitive ratio of online algorithm ON on this particular instance. Note that $p_n = 2^{n-1} \cdot p_1$ and $\sum_{i=1}^{n} p_i = (2^n - 1)p_1$

$$\begin{aligned}
\frac{C_{ON}(\sigma_n)}{C_{OPT}(\sigma_n)} &= \frac{x_0 + p_0 + \sum_{i=1}^{n}(y_i + q_i + x_i + p_i)}{\sum_{i=1}^{n} q_i + p_n} \\
&= \frac{x_0 + p_0 + (2^n - 1) \cdot p_1 + \sum_{i=1}^{n}(y_i + q_i + x_i)}{\sum_{i=1}^{n} q_i + 2^{n-1} \cdot p_1} \\
&= 2 - \frac{\sum_{i=1}^{n} q_i - \sum_{i=1}^{n}(y_i + x_i) - x_0 - p_0 + p_1}{\sum_{i=1}^{n} q_i + 2^{n-1} \cdot p_1}
\end{aligned}$$

Secondly, we prove that:

$$\frac{\sum_{i=1}^{n} q_i - \sum_{i=1}^{n}(y_i + x_i) - x_0 - p_0 + p_1}{\sum_{i=1}^{n} q_i + 2^{n-1} \cdot p_1} \to 0$$

as n goes to infinity for any online algorithm with competitive ratio strictly less than 2. However, this is a contradiction with the competitive ratio being strictly less than 2. As a result, we have proven our main theorem:

Theorem 1 *No online algorithm for $P2|online-list, m_j|C_{max}$ has a competitive ratio strictly less than 2.* □

3 Conclusions and Remarks

We have shown that there does not exist a online algorithm for $P2|online - list, m_j|C_{max}$ with competitive ratio strictly smaller than 2. Thereby, we not only improve the existing lower bound on the competitive ratio of $1 + \sqrt{\frac{2}{3}}$, but also close the gap with the upper bound.

Although, greedy is the best possible in the two machine case, it is certainly not for the case with m machines. With m machines an greedy algorithm has competitive ratio m, while the best known upper bound on the competitive ratio for an arbitrary number of machines is 8. For future research it would be interesting to improve both the lower and the upper bounds of the competitive ratio for the case with m machines.

References

[1] B. Johannes (2006), Scheduling parallel jobs to minimize the makespan, *Journal of Scheduling* **9**, 433 – 452.

[2] D. Ye and G. Zhang (2004), On-line Scheduling of Parallel Jobs, *Lecture Notes in Computer Science* **3104**, 279 – 290.

[3] W.T. Chan, F.Y.L. Chin, D. Ye, G. Zhang, and Y. Zhang (to appear), On-Line Scheduling of Parallel Jobs on Two Machines, *Journal of Discrete Algorithms*.

Multi-skill Project Scheduling Problem and Total Productive Maintenance

Cédric Pessan
LI, Université François Rabelais Tours, 64 av. Jean Portalis 37200 Tours, France, cedric.pessan@univ-tours.fr
SKF France SA, Industrial Division, MDGBB Factory, 204, boulevard du Charles de Gaulle 37542 Saint-Cyr-sur-Loire CEDEX, France

Odile Bellenguez-Morineau, Emmanuel Néron
LI, Université François Rabelais Tours, 64 av. Jean Portalis 37200 Tours, France

In this paper, we propose a multi-skill project scheduling problem model for maintenance activities organization. Preventive maintenance activities are usually planned in advance: production is stopped and all maintenance activities should be processed as fast as possible in order to restart production. Moreover, these human resource handled activities require specific skills and are subject to precedence constraints. The main difference with Multi-Skill Project Scheduling Problem is that some activities may be submitted to disjunctive constraints due to material constraints of the production channel that we consider. We describe how we use these constraints to improve usual MSPSP resolution methods.

Keywords: Applications, Multi-Skill Project Scheduling Problem, Maintenance

1 Introduction

This study, based on a real industrial case found in MDGBB[1] SKF factories, deals with the practical organization of Total Productive Maintenance activities. Total Productive Maintenance (also known as TPM [7]) is a maintenance concept that includes scheduled maintenance downtime of production channels. The idea is to achieve all preventive tasks during one single period. A preventive task is something that has to be done periodically on the machines of the channel. For example, if a filter has to be changed on a machine every 3 months, it may be interesting to change it every 2 months during the scheduled downtime so that there is no specific unscheduled downtime due to this filter. Considering there is a large number of such tasks, the productivity of the channel can be greatly improved if one single downtime period can be scheduled during which all maintenance activities can be done. In this paper, we study how the tasks planned for a given maintenance downtime can be scheduled in order to restart production as soon as possible. Here, we consider that the number of tasks, their duration and the resources they require are given.

We describe how this problem can be modeled as a Multi-Skill Project Scheduling Problem and how exact method can be adapted for solving TPM problem. Finally, we show how the disjunctive constraints that are due to material constraints of the production channel are modeled and how they can be used to improve the efficiency of the exact method.

2 Problem definition

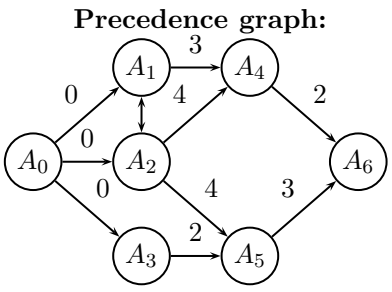
Each maintenance task is an activity $A_i, i \in \{1, \ldots, n\}$ with n being the number of activities. These activities may be submitted to precedence constraints, so an activity cannot start before all its predecessors are completed. In order to perform these activities, M staff members $P_m, m \in \{1, \ldots, M\}$

[1] Medium Deep Groove Ball Bearings

are available. To be processed, activities require specific skills that can not be performed by all staff members. For instance, an electronics specialist would not be able to perform a mechanical operation. The maintenance activities require K skills named $S_k, k \in \{1, \ldots, K\}$ and we define $S_{m,k} = 1$ if P_m masters the skill S_k and 0 otherwise. In the SKF case, there exist hierarchical levels of skills: it means that a person with level x for a skill S_k will not be able to perform a task that requires this skill but at a higher level $y > x$. This problem has been shown to be equivalent to the general problem without hierarchical levels [2], by adding a specific skill corresponding to each level. Lastly, we define the skill requirements for each activity: $b_{i,k}, i \in \{1, \ldots, n\}, k \in \{0, \ldots, K\}$ the number of persons who masters the skill S_k required to perform A_i. We assume that a staff members is able to perform at most one skill for one activity at a time.

The main difference between the MSPSP and TPM problem is the existence of additional disjunctive constraints. These constraints are introduced for security reasons, due to material configuration of the production channel: for instance, when someone is working on a specific machine, it may be avoided having another technician doing something else on the same machine. So, these are disjunctive constraints that are not related to precedence constraints. A simple way to model these disjunctive constraints is to add a common resource requirement for all disjunctive activities, corresponding for instance to the machine on which these activities are performed. For example, if A_i, A_j, A_k cannot be performed at the same time, we add a skill $S_{dis_{i,j,k}}$ that can only be performed by a virtual person $P_{dis_{i,j,k}}$. This person only has this skill and the activities are the only ones to require this resource. Adding these new requirements is enough to take into consideration the disjunctive constraints, in the MSPSP model.

Here is presented a problem instance : disjunctive/conjunctive graph used to model precedence and disjunctions, skill requirements of activities and their processing times. On this example specific skill ($S_{dis_{1_2}}$), and specific resource ($P_{dis_{1_2}}$) corresponding to disjunction ($A_1 - A_2$) have been added to the initial instance.



Skill requirements:

$b_{i,k}$	S_0	S_1	S_2	$S_{dis_{1,2}}$	p_i
A_1	2	1	-	1	3
A_2	-	-	3	1	4
A_3	-	1	-	-	2
A_4	1	-	1	-	2
A_5	-	1	1	-	3

Skills:

$MS_{m,k}$	S_0	S_1	S_2	$S_{dis_{1,2}}$
P_0	1	-	1	-
P_1	1	-	-	-
P_2	1	1	1	-
P_3	-	1	1	-
$P_{dis_{1,2}}$	-	-	-	1

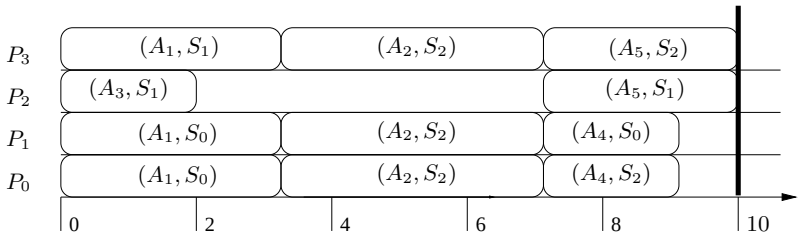


Figure 1: Example of Total productive maintenance problem

3 Solving the Total Maintenance Problem

To solve this problem, we have used an existing Branch-and-Bound method described in [3] for solving MSPSP. The branching scheme of this method is based on splitting time-windows of activities, inspired from schedule scheme proposed by Carlier and Latapie [5] for solving RCPSP. At each node, a time-window is reduced until all activities have their starting time fixed. The Branch-and-Bound method, or MSPSP, uses two lower bounds. The first one is based on a compatibility graph: it checks which activities can be processed at the same time. The second lower bound is an adaptation of energetic reasoning [1] that checks if the mandatory parts of the activities during an interval can be processed on the available resources, respecting skill constraints. The computation of the mandatory parts is more efficient if the time-windows are smaller so this lower bound can also be improved by taking into consideration disjunctive constraints. Disjunctive constraints are used both to compute lower bounds and to reduced time-bounds of activities, using for instance edge-finding methods [6]. Reducing time windows is important both for reducing size of the search tree and improving efficiency of the lower bound. Moreover, based on the work of Baptiste et al. [1] for the RCPSP, we apply methods, e.g., [4] and [6], for selecting disjunctive constraints, on the top of the search tree.

4 Conclusion

In this paper, we have presented how the different activities involved during a scheduled maintenance period of a production channel can be optimized using a Multi Skill Project Scheduling Problem resolution method. This problem is made up of disjunctive constraints that can be used to reduce the search space of a Branch-and-Bound method. More details along with experimental results will be presented during the conference.

References

[1] P. Baptiste and C. Le Pape (2000), Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems, *Constraints* **5**, 119 – 139.

[2] O. Bellenguez and E. Néron (2005), Lower bounds for the Multi-Skill Project Scheduling Problem with hierarchical levels of skills, *Lecture notes in computer science 3616/2005 PATAT 2004, Revised Selected Papers*, 229 – 243.

[3] O. Bellenguez-Morineau (2006), *Méthodes de résolution pour un problème de gestion de projet multi-compétence*, PhD Thesis, Université François Rabelais Tours.

[4] J. Carlier and E. Pinson (1990), A practical use of Jackson's preemptive schedule for solving the Job-Shop problem, *Annals of Operations Research* **26**, 269 – 287.

[5] J. Carlier J. and B. Latapie (1991), Une mthode arborescente pour rsoudre les problmes cumulatifs, RAIRO-RO, **25**(3), 311–340.

[6] J. Carlier and E. Pinson (1994), Adjustment of heads and tails for the job-shop problem, *European Journal of Operational Research* **78**(2), 146 – 161.

[7] S. Nakajima (1988), *Introduction to Total Productive Maintenance*, MA:Productivity Press, Cambridge.

A Methodology for Integrated Risk Management and Proactive Scheduling of Construction Projects

Damien Schatteman, Willy S. Herroelen, Stijn Van de Vonder
Katholieke Universiteit Leuven, Department of Decision Sciences and Information Management, Belgium,
{Damien.Schatteman, Willy.Herroelen, Stijn.VandeVonder}@econ.kuleuven.be

Anton Boone
Belgian Building Research Institute (BBRI), Belgium, anton.boone@bbri.be

In demand of and in cooperation with the Belgium Building Research Institute we developed a methodology that helps to build, through risk assessment and proactive/reactive scheduling, a robust project schedule. The methodology is under test on real life construction projects. We document such an application in the last section.

---

1 Introduction

Construction projects have to be performed in complex dynamic environments that are often characterized by uncertainty and risks. The literature contains ample evidence that many construction projects fail to achieve their time, budget and quality goals ([1],[2]). A study by Maes [10] revealed that inferior planning was the third major cause of company bankruptcies in the Belgian construction industry.

We describe a methodology for risk assessment and proactive/reactive construction project scheduling. Risk management in the construction industry has mostly been used for measuring the impact of potential risks on global project parameters such as time and costs. The literature provides both fuzzy approaches, mixed quantitative/qualitative assessment and risk response methods ([3], [4], [11]). Unlike these approaches, we rely on an integrated methodology that not only allows for uncertainty estimation at the level of the individual project activities, but also uses this input for a proactive scheduling system to generate a robust baseline schedule that is sufficiently protected against anticipated disruptions that may occur during project execution while still guaranteeing a satisfactory project makespan performance.

The methodology relies on a computer supported risk management system that uses a graphical user interface to support project management in the identification, analysis and quantification of the major project risk factors and to derive the probability of their occurrence as well as their impact on the duration of the project activities. Using estimates on the marginal cost of activity starting time disruptions provided by project management, a buffer insertion algorithm is used to generate a proactive baseline schedule that is sufficiently protected against anticipated disruptions that may occur during project execution without compromising on due date performance.

2 Risk assessment

In order to anticipate the risks involved with construction projects it is necessary to quantify them. In practice, software packages ask the project managers to quantify each activity independently. If it is a large project, this is a gigantic work. To minimize this workload we propose a new approach.

We minimize the workload by grouping activities with similar risk profiles. This grouping simplifies the subsequent risk analysis process, which can now be performed at the activity group level rather than at the level of each individual project activity. Secondly, we identify and quantify

the risks on activity group level. The quantification approach is somewhat similar to the *minimalist first pass approach* of Chapman and Ward [5]. Similar to theirs, our approach expects project management to provide the probability of occurrence and the impact of the risk factors on the activities of an activity group under a best case scenario and a worst case scenario. This results in probability density functions for each risk associated with an activity group. Finally, the probability density functions of the impacts for all detected risks are projected on the individual project activities thus yielding the activity duration distribution functions (Triangular or Beta).

This projection is the key to the reusability of the risk assessments at the level of the activity groups. Using the characteristics of the risk impact densities and probability, risk assessments and/or risk data coming from project records and distributions for project activities can be calculated and entered in a risk management database.

Other than the duration distributions, the activity weights are defined. The weights reflect the scheduling flexibility of the activities in the groups and will be used by the robust project scheduling procedures. For the technical details of the approach we refer to Schatteman et. al. ([9]).

3 The robust project scheduling system

The robust project scheduling system we propose relies on the generation of a robust project baseline schedule that anticipates identified risks and that is sufficiently protected against distortions that may occur during actual project execution. The system takes as input the data generated during the risk identification and quantification process described in the previous section.

The robust baseline schedule is generated by introducing time buffers in a precedence and resource feasible project schedule. Time buffering is one of the possible techniques for generating proactive project schedules ([7],[12],[6]).

We use the bi-criteria objective of maximizing the timely project completion probability and minimizing the weighted sum of the expected absolute deviation in activity starting times.

Excellent results have been obtained by the *Starting Time Criticality* (STC) heuristic, that exploits the information generated by the risk assessment procedure described earlier.

4 Applying the framework to a real-life project

We document the application of our risk management and proactive/reactive scheduling framework to a real-life project in the Belgian construction industry. The housing project involved the construction of a five-story apartment building in Brussels.

We used the initial project network developed by the project team and their activity time estimates as input. During the risk assessment procedure, use could be made of the risk management database maintained at the BBRI, which contained risk data obtained on a similar construction project.

Four procedures were used to construct a baseline schedule.

In a first analysis we take the four generated baseline schedules as input and submit them to disruptions in simulation runs of the project, using the estimated activity distributions and imposing a reactive scheduling procedure at schedule breakage that applies a robust parallel generation scheme based on a priority list that orders the activities in non-decreasing order of their starting times in the baseline schedule ([12]). The requested service level for each of the four schedules is set to 99%.

A second analysis is performed upon completion of the project. At the time when the actual disturbances that occurred during the execution of the project were all known, we confronted the

four baseline schedules with the actual schedule disruptions. Each time a schedule was disrupted, the same reactive procedure as used in the first analysis, was used to repair the schedule.

The validation of the procedures revealed that the schedule generated by MS Project® sets a complete unrealistic planned project delivery date, which was violated by more than four months due to numerous schedule breakages. The schedule generated by ProChain® using a 50 % buffer sizing rule, included too much protection and was subjected to numerous disruptions, resulting into high instability costs. Using the sum of squares buffer sizing rule does away with the unacceptable large target buffers but was still outperformed by the STC schedule on both makespan performance and stability cost.

References

[1] J. Al-Bahar and K. Crandal (1990), Systematic risk management approach for construction projects, *Journal of Construction Engineering and Management* **116**, 533 – 546.

[2] S. Assaf and S. Al-Hejji (2006), Causes of delay in large construction projects, *International Journal of Project Management* **24**, 349 – 357.

[3] I. Ben-David and T. Raz (2001), An integrated approach for risk response development in project planning, *Journal of the Operational Research Society* **52**, 14 – 25.

[4] V. Carr and J. Tah (2001), A fuzzy approach to construction project risk assessment and analysis: construction project risk management system., *Advances in Engineering Software* **18**, 369 – 383.

[5] C. Chapman and S. Ward (2000), Estimation and evaluation of uncertainty: A minimalist first pass approach., *International Journal of Project Management* **18**, 369 – 383.

[6] E. Goldratt (1997), *Critical Chain*, The North River Press Publishing Corporation, Great Barrington.

[7] W. Herroelen and R. Leus (2004), Robust and reactive project scheduling: a review and classification of procedures., *International Journal of production Research* **42**, 1599 – 1620.

[8] J. Maes, C. Vandoren, L. Sels and F. Roodhooft (2006), Onderzoek naar oorzaken van faillissementen van kleine en middelgrote bouwondernemeingen., Research report, Departement of applied economics, Katholieke Universiteit Leuven, Belgium.

[9] D. Schatteman , W.S. Herroelen WS, S. Van De Vonder and A. Boone (2006), A methodology for integrated risk management and proactive scheduling of construction projects., FETEW Research Report KBI **0622**, Katholieke Universiteit Leuven, Belgium.

[10] S. Van de Vonder, E. Demeulemeester, W. Herroelen and R. Leus (2005), The trade-off between stability and makespan in resource-constrained project scheduling, *International Journal of Production Research* **44**, 215 – 236.

[11] S. Van de Vonder (2006), Proactive/reactive procedures for robust project scheduling, *PhD thesis*, Research Center for Operations Management, Katholieke Universiteit Leuven, Belgium.

[12] A. Warszawski and R. Sacks (2004), Practical multifactor approach to evaluating risk of investment in engineering projects., *Journal of Construction Engineering and Management* **130**, 357 – 367.

An Exact Algorithm for Single-Machine Scheduling without Idle Time

Shunji Tanaka
Kyoto University, Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan, tanaka@kuee.kyoto-u.ac.jp

This study proposes an exact algorithm for the single-machine scheduling problem to minimize the sum of job completion costs, where machine idle time is prohibited. This algorithm is based on the the SSDP (Successive Sublimation Dynamic Programming) method, which starts from a relaxation of the original problem and then successively solves relaxations with more detailed information by dynamic programming. In this study several improvements are proposed to reduce both the memory requirement and the computational time. Numerical experiments show that the proposed algorithm can handle 300 jobs instances of the total weighted tardiness problem and 200 jobs instances of the total weighted earliness and tardiness problem.

1 Introduction

This study treats a class of scheduling problems to sequence jobs on a single machine so that the sum of job completion costs is minimized. More specifically, we consider that a set of n jobs $\mathcal{N} = \{1, \ldots, n\}$ is to be processed on a single machine without preemption. Job i ($i \in \mathcal{N}$) is given an integer processing time p_i and an integer cost function $f_i(t)$. Machine idle time is not permitted and the machine cannot process more than one job at a time. Hence, all the jobs should be processed in the interval $[0,\ T]$, where $T = \sum_{i=1}^{n} p_i$. The objective is to find an optimal job sequence to minimize $\sum_{i=1}^{n} f_i(C_i)$, where C_i denotes the completion time of job i.

For this problem, an efficient lower bound computation method based on the state-space relaxation technique [1] was proposed by Abdul-Razaq and Potts [2]. Based on their results, Ibaraki and Nakamura [3] applied the successive sublimation dynamic programming (SSDP) method [4] to this problem. In this algorithm, the constraint on successive jobs is first added to the dynamic programming state-space relaxation of the original problem (it is equivalent to the Lagrangian relaxation of the constraint on the number of job occurrences). Then, state-space modifiers are successively added until the duality gap becomes zero. In the course of the algorithm, dynamic programming states are eliminated to reduce the memory requirement by computing lower bounds for passing through the states. Nevertheless, they concluded that their algorithm is hard to apply due to the heavy memory requirement when the scheduling horizon T becomes large.

The purpose of this study is to construct an exact algorithm based on the SSDP method. There are several improvements from the original algorithm by Ibaraki and Nakamura [3]:

(1) lower bound improvement and state elimination by the dominance of adjacent jobs [5],

(2) efficient upper bound computing by the iterated enhanced dynasearch [6, 7],

(3) sophisticated step sizing in subgradient optimization,

(4) improved selection of state-space modifiers,

(5) further state elimination by the dominance of four successive jobs.

These improvements enable us to reduce both the memory requirement and the computational time, and the algorithm can handle the total weighted tardiness problem ($1||\sum w_i T_i$) instances

with up to 300 jobs, and the total weighted earliness and tardiness problem ($1||\sum(\alpha_i E_i + \beta_i T_i)$) instances with up to 200 jobs, while our implementation of the original algorithm could not solve some of the 40 jobs instances optimally.

2 Outline of the algorithm

Stage 1 The initial upper bound is computed by applying the iterated enhanced dynasearch [6, 7] to an SPT sequence. Then, subgradient optimization is applied to a Lagrangian dual corresponding to a Lagrangian relaxation of the original problem. In this relaxation, the constraint on the number of job occurrences is relaxed and the constraint that no job duplication occurs in any *two* successive jobs [2] is added. State elimination is performed by applying both forward and backward dynamic programmings every time when the lower bound is improved. After the subgradient optimization is terminated, the upper bound is updated by applying the iterated enhanced dynasearch to a solution constructed from a solution of the relaxation.

Stage 2 Starting from the multipliers obtained in Stage 1, subgradient optimization is applied to a Lagrangian dual corresponding to a more constrained relaxation of the original problem. In this relaxation, the constraint that no job duplication occurs in any *three* successive jobs [2, 3, 8] and another constraint on the dominance of adjacent jobs [5] are added. An upper bound is also computed in every five iterations, and state elimination is performed every time when either the lower bound or the upper bound is improved.

Stage 3 While state-space modifiers are successively added, the relaxation in Stage 2 is solved in forward or backward manner by turns until the current upper bound is proved to be optimal. The upper bound computation, state elimination, and further state elimination based on the dominance of four successive jobs are applied every time when the relaxation is solved. Modifiers are added to maximum two jobs separately so that they should appear exactly once. These jobs are chosen from the jobs that never appear and then that appear most frequently in the solution of the relaxation. Ties are broken by the smallest number of occurrences in the current dynamic programming state space to suppress the increase of states.

3 Numerical experiments

With regard to $1||\sum w_j T_j$, the instances in [9] are used as 40, 50 and 100 jobs instances ($n = 40, 50, 100$) (available from OR-Library: http://people.brunel.ac.uk/~mastjjb/jeb/info.html). The instances with $n = 150, 200, 250, 300$ are generated in a similar way to the OR-Library instances. Thus 125 problem instances are considered for each n. The instances of $1||\sum(\alpha_i E_i + \beta_i T_i)$ are generated from those of $1||\sum w_j T_j$, where integer earliness weights are generated by the uniform distribution in $[1, 10]$. Computation is performed on a Pentium4 2.4GHz desktop computer with 512MB RAM by running a code written in C (gcc). The maximum memory size for dynamic programming states is restricted to 384MB.

The results are shown in Table 1. Only one instance of $1||\sum(\alpha_i E_i+\beta_i T_i)$ with 200 jobs cannot be solved optimally due to the heavy memory requirement. Nevertheless, we can see that the proposed algorithm can handle 300 jobs instances of $1||\sum w_j T_j$ and 200 jobs instances of $1||\sum(\alpha_i E_i + \beta_i T_i)$. Compared to the algorithm in [10] for $1||\sum w_j T_j$ and the algorithm in [11] for $1||\sum(\alpha_i E_i + \beta_i T_i)$, our algorithm seems much faster because the algorithm in [10] requires maximum 9 hours on a Pentium4 2.8GHz computer to solve the OR-Library 100 jobs instances and the algorithm in [11] could not solve some of 40 jobs instances within 3600 seconds on a Pentium II 266MHz computer.

Table 1: Average and maximum CPU times of the proposed algorithm

n	$1\|\|\sum w_iT_i$			$1\|\|\sum(\alpha_iE_i+\beta_iT_i)$		
	CPU time (s)		solved	CPU time (s)		solved
	average	maximum		average	maximum	
40	0.20	0.99	125/125	0.25	0.52	125/125
50	0.42	0.98	125/125	0.58	1.87	125/125
100	7.16	50.04	125/125	12.01	27.69	125/125
150	30.93	213.76	125/125	68.55	285.29	125/125
200	82.21	355.60	125/125	*270.27**	*2032.58**	124/125
250	200.68	1534.78	125/125			
300	446.52	5456.41	125/125			

*The average and maximum are taken over optimally solved instances.

Moreover, the framework of this study is more general and it can be applied to any single-machine machine scheduling problems without idle time to minimize the sum of job completion costs.

Acknowledgement: This work has been partially supported by Grant-in-Aid for Young Scientists (B) 19760273, from Japan Society for the Promotion of Science (JSPS).

References

[1] N. Christofides, A. Mingozzi and P. Toth (1981), State-space relaxation procedures for the computation of bounds to routing problems, *Networks* **11**, 145 – 164.

[2] T.S. Abdul-Razaq and C.N. Potts (1988), Dynamic programming state-space relaxation for single-machine scheduling. *J. Oper. Res. Soc.* **39**, 141 – 152.

[3] T. Ibaraki and Y. Nakamura (1994), A dynamic programming method for single machine scheduling, *Eur. J. Oper. Res.* **76**, 72 – 82.

[4] T. Ibaraki, (1987), Enumerative approaches to combinatorial optimization, *Ann. Oper. Res.* **10** and **11**.

[5] S. Tanaka, S. Fujikuma and M. Araki (2006), A branch-and-bound algorithm based on Lagrangian relaxation for single-machine scheduling, *International Symposium on Scheduling 2006*, 148 – 153.

[6] R.K. Congram, C.N. Potts and S.L. van de Verde (2002), An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, *INFORMS J. Comput.* **14**, 52 – 67.

[7] A. Grosso, F. Della Croce and R. Tadei (2004), An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem *Oper. Res. Lett.* **32**, 68 – 72.

[8] L. Péridy, É. Pinson and D. Rivreau (2003), Using short-term memory to minimize the weighted number of late jobs on a single machine, *Eur. J. Oper. Res.* **148**, 591 – 603.

[9] H.A.J. Crauwels, C.N. Potts and L.N. Van Wassenhove (1998), Local search heuristics for the single machine total weighted tardiness scheduling problem, *INFORMS J. Comput.* **10**, 341 – 350.

[10] Y. Pan and L. Shi (2007), On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems, *Math. Prog. A* **110**, 543 – 559.

[11] C.-F. Liaw (1999), A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem, *Comput. Oper. Res.* **26**, 679 – 693.

An Efficient Heuristic Procedure for the Resource Availability Cost Problem

Vincent Van Peteghem
Ghent University, Tweekerkenstraat 2, B-9000 Gent, Belgium, vincent.vanpeteghem@ugent.be
Mario Vanhoucke
Ghent University, Tweekerkenstraat 2, B-9000 Gent, Belgium, mario.vanhoucke@ugent.be
Vlerick Leuven Gent Management School, Reep 1, B-9000 Gent, Belgium, mario.vanhoucke@vlerick.be

In this paper, we study the resource availability cost problem. This problem description minimizes the total cost of the (unlimited) renewable resources required to complete the project by a pre-specified project deadline. We developed a heuristic procedure to solve the RACP starting from a heuristic upper bound solution and searching for iterative gradual improvements in the total resource cost.
Keywords: Project management; Resource availability cost; Heuristics; Scatter Search

1. Introduction

Resource constrained project scheduling has been a research topic for many decades, resulting in a wide variety of optimization procedures. The main focus on project lead time minimization has led to the development of various exact and (meta-)heuristic procedures for scheduling projects with tight resource constraints under a wide variety of assumptions. The basic problem type in project scheduling is the well-known resource-constrained project scheduling problem (RCPSP). This problem type aims at minimizing the total duration or makespan of a project subject to precendence relations between the activities and the limited renewable resource availabilities, and is known to be NP hard [1]. In the last few decades, several effective exact and heuristic algorithms for solving the RCPSP have been proposed. For an overview of resource-constrained project scheduling in general, we refer to excellent overview papers of [2], [8], [9], [10] and [12]. Less attention, however, has been given to its close variant, the resource availability cost problem (RACP). This problem description minimizes the total costof the (unlimited) renewable resources required to complete the project by a pre-specified project deadline.

In this paper, we study the resource availability cost problem, denoted as $m,1|cpm,\delta_n|rac$ using the classification scheme of Herroelen et al. [8]. The RACP can be stated as follows. A set of activities N, numbered from a dummy start node 0 to a dummy end node $n + 1$, is to be scheduled without pre-emption on a set R of renewable resource types. Each activity $i \in N$ has a deterministic duration d_i and requires r_{ik} units of resource type $k \in R$ which has a constant availability a_k throughout the project horizon. We present a project network in an activity-on-the-node format where A is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists, and a dummy start node 0 and end node $n + 1$ representing the start and completion of the project. These dummy nodes have zero duration while the other activities have a non-zero duration; the dummies also have zero resource usage. We assume graph $G(N,A)$ to be acyclic.

A schedule S is defined by a vector of activity start times and is said to be feasible if all precedence and renewable resource constraints are satisfied. The objective of the RACP is to find a feasible schedule within a pre-specified project deadline δ_{n+1} such that the total resource cost is minimised. We define $C_k(a_k)$ as a discrete non-decreasing cost function associated with the availability a_k of the resource type k. The variables are the resource availabilty values a_k and the start times s_i of each project activity i.

The RACP can be conceptually formulated as follows:

$$\text{Minimize } \sum_{k \in R} C_k(a_k) \qquad [1]$$

Subject to

$$s_i + d_i \le s_j \qquad \forall (i, j) \in A \qquad [2]$$

$$\sum_{i \in S(t)} r_{ik} \le a_k \qquad \forall k \in R \text{ and } t = 1, \ldots, \delta_{n+1} \qquad [3]$$

$$s_0 = 0 \qquad [4]$$

$$s_{n+1} \le \delta_{n+1} \qquad [5]$$

Eq. [1] minimizes the total resource cost of the project. Eq. [2] takes the finish-start precedence relations with a time-lag of zero into account. The renewable resource constraints are satisfied thanks to eq. [3], where $S(t)$ represent the set of activities in progress during the time interval $]t - 1, t]$. Eq. [4] imposes a pre-specified deadline to the project and Eq. [5] forces the project to start at time instance zero.

2. Solution procedure

2.1 Solution methods from literature

This problem was introduced by Möhring [11] as the resource investment problem. He proposes an exact procedure based on graph theoretical algorithms for comparability graph and interval graph recognition and orientation.

Demeulemeester [4] developed an effective optimal algorithm, based on iterative solutions for the RCPSP found by the branch-and-bound algorithm of Demeulemeester and Herroelen [3]. The search makes use of so-called efficient points which delimit the solution space of all possible combinations of resource availabilities. The algorithm tries to schedule the problem based on the cheapest efficient point. If the makespan is larger than the deadline, the availability of a single resource type is increased by one unit. The procedure is repeated until the makespan is smaller than or equal to the deadline. In order to save computational time, the algorithm is truncated after a prespecified number of iterations in each branch-and-bound tree, and hence, the procedure is mainly used as a heuristic search procedure.

Drexl and Kimms [6] propose two lower bounds for the RACP using Lagrangean relaxation and column generation techniques. Yamashita et al. [14] developed a multi-start heuristics based on the scatter search methodology. An improvement heuristic is proposed which increases the resource availabilities of infeasible schedules based on the start time of the activities as well as on their latest starting times, in order to make the makespan equal to the deadline. Yamashita et al. [14] also propose a multi-start heuristic for projects with uncertain activity durations.

2.2 Our solution method

We developed a heuristic procedure to solve the RACP starting from a heuristic upper bound solution and searching for iterative gradual improvements in the total resource cost.

The iterative search for gradual improvements makes use of random key values which are transformed into resource feasible schedules based on an extended version of the serial schedule generation scheme. The search to random keys is based on an enumeration scheme which iteratively reduces the total resource cost of the project. The adapted serial schedule generation scheme relies on the philosophy of the efficient points, and iteratively solves a limited number of RCPSP instances per random key.

We compare our new solution approach with the heuristic procedure of [14], with the efficient procedure of [4] and with straightforward extensions of the latter procedure where the RCPSP is solved by meta-heuristic RCPSP solution approaches rather than a truncated branch-and bound procedure.

3. Computational results

We have coded our algorithm in Visual C++ 6.0 and tested and validated it on a dataset randomy generated by RanGen [5] containing project scheduling instances with up to 100 project activities and 6 renewable resource types.

We study the effect of the topological structure of a project network, the resource scarceness as well as the cost structure of various resource types on the solution quality. We also test the influence of various algorithmic parameters on the solution quality and the required CPU time. Preliminary results will be presented on the MISTA 2007 workshop.

References

[1] Blazewicz, J., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, **5**, 11 – 24.

[2] Brücker, P., A. Drexl, R. Möhring, K. Neumann, E. Pesh (1999), Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* **112**, 3 – 41.

[3] Demeulemeester, E. and W. Herroelen (1992), A Branch-and-Bound procedure for the Multiple Resource-Contrained Project Scheduling Problem, *Management Science* **21**, 944 – 955.

[4] Demeulemeester, E. (1995), Minimizing Resource Availability Costs in Time-Limited Project Networks, *Management Science* **41**, 1590 – 1598.

[5] Demeulemeester, E., M. Vanhoucke and W. Herroelen (2003), RanGen: A random generator for activity-on-the-node networks, *Journal of Scheduling* **6**, 17 – 38.

[6] Drexl, A. and A. Kimms (2001), Optimization guided lower and upper bounds for the resource investment problem, *Journal of the Operational Research Society* **52**, 340 – 351.

[7] Herroelen, W., B. De Reyck and E. Demeulemeester (1998), Resource-constrained project scheduling: A survey of recent developments, *Computers & Operations Research* **25**, 279 – 302.

[8] Herroelen, W., B. De Reyck and E. Demeulemeester (1999), *A Classification Scheme for Project Scheduling, Chapter 1* in Weglarz, J. (ed.), Handbook of Recent Advances in Project Scheduling, Kluwer Academic Publishers, 1 – 26.

[9] Icmeli, O, S. Selcuk Erenguc and C.J. Zappe (1993), Project Scheduling Problems: A Survey, *International Journal of Operations and Production Management* **13**, 80 – 92.

[10] Kolisch, R. and Padman, R. (2001), An integrated survey of deterministic project scheduling, *Omega* **49**, 249 – 272.

[11] Möhring, R.H. (1984), Minimizing costs of resource requirements in project networks subject to a fixed completion time, *Operation Research* **32**, 89 – 120.

[12] Özdamar, L. and Ulusoy, G. (1995), A survey on the resource-constrained project scheduling problem, *IIE Transactions* **27**, 574 – 586.

[13] Yamashita, D.S., V.A. Armentano and M. Laguna (2006), Scatter Search for project scheduling with resource availability cost, *European Journal of Operational Research* **169**, 623 – 637.

[14] Yamashita, D.S., V.A. Armentano and M. Laguna (2007), Robust optimization models for project scheduling with resource availability cost, *Journal of Scheduling* **10**, 67 – 76.

A Strengthened Continuous Time Formulation for the Cyclic Scheduling of a Plant

Francois Warichet, Yves Pochet
CORE-UCL, 34 Voie du Roman Pays, 1348 Louvain-la-Neuve, Belgium, {warichet, pochet}@core.ucl.ac.be

We study in this paper a continuous time formulation for the cyclic scheduling of mixed production lines, i.e. composed of batch and continuous processes. The objective is to maximize productivity. We base our formulation on the resource task network representation proposed by Pantelides [5]. The initial continuous time mixed integer linear programming (MILP) formulation proposed is similar to the one introduced by Schilling and Pantelides [6]. Other continuous time formulations were proposed by Castro et al. [2] and Wu and Ierapetritou [9].

The main characteristic of this continuous time formulation is that the size of the time interval between two events (i.e points in time when the system state changes) and the duration of the cycle are variables of the model. Moreover, this continuous time formulation is very compact in the sense that the number of events is much smaller than the number of time periods needed for the corresponding discrete time formulation. The latter formulation is usually quite tight but the size of the time period is fixed and has to be smaller than the greatest common divisor of all the processing times of the batch processes. Therefore, the number of time periods needed in discrete time formulations is typically very large, see Shah et al. [7]. Although much more compact, the drawback of continuous time formulations is that they are very weak, i.e. duality gaps are large, because big M type constraints need to be introduced to build a correct model formulation.

We consider a general production process where the resources are the processing units, the utilities shared by the tasks, and the storage tanks containing the intermediate products produced or consumed by the tasks. In this process, there are both batch and continuous tasks. Each batch task has a fixed processing time, can be processed on a subset of reactors and can be repeated several times. Precedence and zero waiting time constraints exist between some of the batch tasks. Each continuous task has a limited process rate. The batch and the continuous tasks consume and produce resources, for which we have some capacity restrictions. Moreover, some of the continuous tasks cannot be interrupted. The objective is to obtain a cyclic schedule of the mixed plant maximizing its productivity, where productivity is defined as the quantity of finished product produced over one cycle, divided by the cycle duration.

We show in this paper how the initial continuous time formulation can be tightened by analyzing and strengthening the formulation of three special cases of the general cyclic scheduling problem described above. The first special case consists of only one batch task with fixed processing time and no resource restriction. Using PORTA, see T. Christof and A. Loebel [3], to analyze polytopes of small instances, we found some valid inequalities that are facet defining for this subproblem.

We describe now the initial continuous time formulation and the new valid inequalities so obtained.

For the continuous time formulation, the time is decomposed in a finite number of time slots and the duration of every time slot is a variable of the model. The decomposition of time is common to all processing units. An event is defined as the beginning or end of a batch task. A time slot is the time between two events (see Figure 1). Events and time slots are numbered from 1 up to T. In cyclic scheduling, the event at the end of time slot T coincides with the event occurring at the beginning of time slot 1, and is numbered as event 1. A batch task has a fixed processing time and produces at the end a fixed amount of product. We are given a batch task, and a maximum number T of event times. The processing time of the batch task is constant and is given by $p[h]$ and

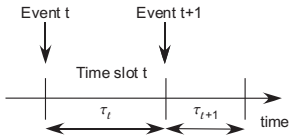


Figure 1: Event and Time slot

t is the index for time slots ($t \in \{1, \ldots, T\}$). The two types of decision variables are the following : τ_t is the duration of time slot t $[h]$ and $z_{t,t'}$ equals 1 if the batch of task starts at the beginning of time slot t and finishes at the end of time slot t', equals 0 otherwise. The initial continuous time formulation for this first subproblem with only one batch task can be written as follows, using a linearized objective to represent the productivity :

$$\max \sum_{t=1}^{T} \sum_{l=t}^{t+d-1} z_{t,\Omega(l)} - \mu \sum_{t=1}^{T} \tau_t \tag{1}$$

$$st \quad pz_{t,\Omega(l)} - \sum_{k=t}^{l} \tau_{\Omega(k)} \leq 0 \qquad \forall t, l : 1 \leq t \leq T, t \leq l \leq t+d-1 \tag{2}$$

$$\sum_{k=t}^{l} \tau_{\Omega(k)} + p(l-t) z_{t,\Omega(l)} \leq p(l-t+1) \qquad \forall t, l : 1 \leq t \leq T, t \leq l \leq t+d-1 \tag{3}$$

$$\sum_{l=t}^{t+d-1} z_{t,\Omega(l)} \leq 1 \qquad \forall t : 1 \leq t \leq T \tag{4}$$

$$\sum_{t=l-d+1}^{l} z_{\Omega(t),l} \leq 1 \qquad \forall l : 1 \leq l \leq T \tag{5}$$

$$z_{t,\Omega(l)} \in \{0,1\} \ \forall t, l : 1 \leq t \leq T, t \leq l \leq t+d-1, \tau_t \geq 0 \ \forall t : 1 \leq t \leq T \tag{6}$$

where d is defined as the maximum number of consecutive time slots for performing a batch task, i.e. if we start a batch task at time slot t, this batch task has to end before the end of time slot $t+d-1$, and where $\Omega(t)$ is the 'wrap-around' time operator defined in Schilling and Pantelides [6] : $\Omega(t) = t \ \forall t : 1 \leq t \leq T$, $\Omega(t) = \Omega(t-T)$ for $t > T$, and $\Omega(t) = \Omega(t+T)$ for $t < 1$.

The objective function (1) corresponds to a measure of the production rate. The constraints (2)-(3) are the timing constraints and ensure that the duration of the batch task is well respected. The constraints (4)-(5) specify the fact that at most one batch task can begin and finish at each time period.

By generalizing the results obtained for small instances, we prove that constraint (2) can be strengthened as follows

$$p \sum_{k=t|t\neq l}^{l} z_{t,\Omega(k)} + p z_{\Omega(l),\Omega(l)} - \sum_{k=t}^{l} \tau_{\Omega(k)} \leq 0 \qquad \forall t, l : 1 \leq t \leq T, t \leq l \leq t+d-1 \tag{7}$$

$$p \sum_{k=t|t\neq l}^{l} z_{\Omega(k),\Omega(l)} + p z_{t,t} - \sum_{k=t}^{l} \tau_{\Omega(k)} \leq 0 \qquad \forall t, l : 1 \leq t \leq T, t \leq l \leq t+d-1 \tag{8}$$

and that constraint (3) can be strengthened by

$$\sum_{t'=t}^{l} \tau_{\Omega(k)} + p \sum_{k=l-d+1}^{l-1} \min\{l-t, l-k\} z_{k,\Omega(l)} \leq p(l-t+1) \tag{9}$$

$$\sum_{t'=t}^{l} \tau_{\Omega(k)} + p \sum_{k=t+1}^{t+d-1} \min\{l-t, k-t\} z_{t,\Omega(k)} \leq p(l-t+1) \tag{10}$$

for all $t, l : 1 \leq t \leq T, t \leq l \leq t+d-1$.

These inequalities (7)-(10) are valid for the subproblem studied and the validity proof is only based on the fact that constraints (4)-(5) impose that at most one batch task can start at each time slot and one can finish. Moreover, we prove that these inequalities are actually facet defining for the subproblem and that with these inequalities, there is no duality gap for the subproblem anymore.

Then, we generalize the formulation results obtained to a second special case where we consider a set of batch tasks. The formulation is further tightened by using strengthening techniques, see Andersen and Pochet [1].

For the third special case consisting of one batch task, one continuous task and one storage tank with restricted capacity between the batch and the continuous task, we show again how to strengthen the initial continuous time formulation.

By using the corresponding strengthened formulation for each of the three subproblems, we show that we can solve these problem instances with fewer nodes and less CPU solution time than by using the initial continuous time formulation.

Based on the improved formulation obtained for the three special cases of the general cyclic scheduling problem, we have extended the results obtained and made them valid for the general cyclic scheduling problem. The strengthened model formulation obtained for the general case is stronger than the initial one and this implies that solutions of relaxations are closer to integral solutions. Therefore, one can hope to solve, with the strengthened formulation, general cyclic scheduling problem instances with less Branch and Bound nodes and therefore also with less CPU time. We observe in practice that by taking advantage of these improvements, the general cyclic scheduling problems is solved quicker by using the strengthened formulation.

However, in order to solve real size industrial cases, the strengthened continuous time formulation obtained does not provide a good feasible solution quickly. Therefore, we introduce MILP based heuristic methods in order to find a hopefully good feasible solution quicker. MIP heuristic methods are based on model formulations and therefore also take advantage of the improvements that we have obtained by tightening the continuous time formulation. We show on one larger instance of a general test case problem that the proposed heuristic method, a combination of two well known MIP based heuristic methods (Relax and Fix, see Stadtler [8], and Local Branching see Fischetti and Lodi [4]), provides good feasible solutions quickly.

References

[1] K. Andersen and Y. Pochet (2006), Coefficient strengthening: a tool for formulating mixed integer programs, *to be submitted*.

[2] P.M. Castro, A.P. Barbosa-Povoa, and H.A. Matos (2003), Optimal periodic scheduling of batch plants using RTN-based discrete and continuous-time formulations : A case study approach, *Ind. Eng. Chem. Res.* **42**, 3346 – 3360.

[3] T. Christof and A. Loebel (1997), PORTA - a polyhedron representation transformation algorithm, *available via http://www.zib.de/Optimization/Software/Porta/*

[4] M. Fischetti and A. Lodi (2003), Local branching, *Mathematical Programming* **98**, 23 – 48.

[5] C.C. Pantelides (1994), Unified frameworks for the optimal process planning and scheduling, Proceedings on the second conference on foundations of computer aided operations, 253 – 274

[6] G. Schilling, C.C. Pantelides (1999), Optimal periodic scheduling of multipurpose plants, *Computers chem. Engng* **23**, 635 – 655.

[7] N. Shah, C.C. Pantelides, and R.W.H. Sargent (1993), Optimal periodic scheduling of multipurpose batch plants, *Annals of Operations Research* **42**, 193 – 228.

[8] H. Stadtler (2003), Multilevel lot sizing with setup times and multiple constrained resources: Internally rolling schedules with lot-sizing windows, *Operations Research* **51**, 487 – 502.

[9] D. Wu, M. Ierapetritou (2004), Cyclic short-term scheduling of multiproduct batch plants using continuous-time representation, *Computers chem. Engng* **28**, 2271 – 2286.

Assigning Part-time Teachers to Courses via a Stable Marriage Algorithm

Vineet Bist, Hoang Do, Navin Sharma and George M. White
School of Information Technology and Engineering, University of Ottawa, Ottawa K1N 6N5, Canada,
white@site.uottawa.ca

1 Introduction

All over the world, it seems, organizations are attempting to obtain more "output" while simultaneously using less "input". Institutes of higher education are no exception. Operating grants from governments are generally decreasing, and as these generate the majority of the revenue used by educational institutions, they increasingly feel financially constrained. In many jurisdictions, these grants are tied to the number of students in attendance. Thus one way of balancing the books is to increase the number of students enrolled, while simultaneously keeping expenses to the minimum possible.

One of the ways of accomplishing this is by using more part-time teachers. They typically are employed to teach one or more courses. They have no tenure, do no research, and can be hired on short term contracts, typically limited to the length of the course. The management of these part-time teachers is considerably easier than the management of tenured regular staff. They are paid considerably less than regular staff and the resources required to service them are greatly reduced over that required by the alternative regular staff members.

In many places, these part-time teachers have formed unions in an attempt to gain some kind of stability in their lives and some additional revenue. This has lead to the creation of collective agreements (CA) that specify the hiring procedures, the rates of pay, the conditions of work and the procedures to be followed in the case where these rules are deemed not to have been followed with the required rigour. These procedures generally include an adjudication mechanism with powers to levy penalties and enforce compliance with the procedures set out in the CA.

It follows, therefore, that management and union both have an interest in seeing that the procedures of the CA are followed to the letter. It was felt by the authors that an algorithmically driven, formal procedure, implemented in a computer would be of assistance in doing this. There is very little published literature on this subject and almost nothing on the problem of assigning courses to part-time teachers. There is nothing at all about the assignment problem subject to the set of constraints specified by the CA at the authors' institution.

2 The Assignment Problem

The problem to be solved here is to assign members of a set of part-time teachers to the members of a set of courses. No attention is paid by the algorithm to the time or place at which the course is to be given. This information is known by applicants when they apply to teach the course in question. Thus this problem falls into the SA - Staff Allocation (Teacher Assignment) category in the Reis and Oliveira taxonomy [1].

The two most relevant entities are the applicant's *seniority* and *competences* for teaching a particular course. An applicant has a seniority that is based mainly on the number of courses previously taught at the home university *as a part-time teacher* . These courses must have been received with a certain minimum degree of enthusiasm by the students concerned as evidenced by

scores received on a formal course evaluation. To be considered successful, a teacher must have received a rating of at least 2.9/5 in the course concerned for the year concerned. A teacher receives

- 1 point for each three credit course successfully taught.
- 2 points for each term containing at least one course successfully taught.
- up to 8 points for performing certain administrative duties.
- 4 points for receiving a research grant and not teaching.

The second entity is the applicant's competence for teaching a particular course. Each course puts certain demands on a teacher: - previous experience with that course, expertise in the subject domain, language skills and the like. Applicants apply to teach one or more courses and include on the application form the reasons why they would do a good job. Based on this information, the administrator ranks them into one of three categories for each applicable course.

- category A (for a course not taught in 1st or 2nd year). This applicant fulfills *all* the qualifications and *all* the required experiences.
- category A (for a course taught in 1st or 2nd year). This candidate fulfills *some* of the qualifications and *some* of the required experiences and has previously taught courses at least three times with a cumulative average of ≥ 4 in certain aspects of the course evaluations.
- category B (for a course not taught in 1st or 2nd year). This candidate *exceeds* all the qualifications and has *all* the required experiences.
- category B (for a course taught in 1st or 2nd year). This requires that the candidate has all the qualifications and experiences and in addition, has additional qualifications over and above those required. As before, the applicant has previously taught courses at least three times with a cumulative average of ≥ 4 in certain aspects of the course evaluations.
- category C. This category includes persons who have obviously superior competences in their fields and can be considered experts.

There are other factors that can raise or lower the assigned category of a course-applicant. These are specified in the C.A. and are not described here. The categories are assigned by an administrator before the assignments are made.

The actual assignment of courses to part-time teachers is presently done manually as follows:

- a) All applicants for a given course are ranked according to category and then by seniority, category being the most important.
- b) Since a teacher is permitted to teach only up to a certain maximum number of courses in two consecutive terms, an individual must be removed from the ranking if acceptance would violate this limit.
- c) The position is then offered to the applicants in the order in which they appear on the list.

If two or more applicants tie for a course, experience previous to Sept. 1, 1981 may be considered. Seniority is then calculated retroactively, one year at a time, until the tie is broken. If the tie is still not broken, the applicant with the most seniority points for the course in question will be assigned the course.

Although this method cannot result in a tie, the solution obtained is not unique. If the courses are arranged in a different order, the method may lead to a different solution. There will be no ties but the solution may not be the best one available in some sense.

3 The Stable Marriage Algorithm

Problems such as this have been encountered many times in the past and procedures have been developed for finding solutions well before the age of computers. They are frequently called match-

ing problems because they involve creating "matches" between the elements of two disjoint sets. These two sets often correspond to tasks and resources, where it is required to assign one or more resources to a set of tasks.

In 1962 Gale and Shapely[2] published a computer based algorithm for solving problems of this type where the number of elements in both sets is the same. This has become known as the stable marriage algorithm since the example used represented a problem in which a set of men and a set of women were to be matched in such a way that no two man-woman pairs would prefer to swap the way in which they were matched. Each man and each woman in the two sets ranks those in the other set in order of preference. The algorithm then calculates a set of matchings, each element consisting of a man-woman pair. The set of matchings is not unique.

A generalized version of this program occurs when elements of one or both of the sets are allowed to associate with more than one element in the other set. In the case where only one of the sets is allowed to associate with more than one member of the other set, and then only up to a certain limit is known as a form of many-to-one matching or the stable marriage problem with bounded polygamy [3]. This algorithm has also been used to assign medical students to hospitals.

Since each one of the teachers has applied to teach a certain number of courses, the order of these courses can be considered to be a max-min preference ranking by the teacher concerned. The category-seniority list for each teacher can likewise be considered to be a max-min preference list for each course if this list is edited to include only those teachers deemed suitable and then ordered as specified above. With these two preference lists, the many-to-one matching algorithm can be run to generate the list of teacher-course assignments. Since a teacher is allowed to teach more than one course up to the permitted maximum, the algorithm is run with a bounded maximum number of courses for any individual.

4 The Aftermath

The system was tested and shown to both the administration and the union. Neither of these bodies reacted as expected. A discussion of the details of the system, the fine points of the algorithm and the reception of the system will be discussed in the full paper.

References

[1] L.P. Reis and E. Oliveira (2001), A language for specifying complete timetabling problems, *Lecture Notes in Computer Science* **2079**, 322 – 341.

[2] D. Gale and L.S. Shapely (1962), College admissions and the stability of marriage. *Amer. Math. Monthly* **69**, 9 – 15.

[3] M. Baïou and M. Balinski (2000), Many-to-many matching: Stable polyandrous polygamy (or polygamous polyandry). *Discrete Applied Mathematics* **101**, 1 – 12.

Author Index

Aghezzaf, El-Houssaine 426
Alkan, Alpay 394
Andrés, Carlos 175, 439
Angelelli, Enrico 51
Artigues, Christian 557
Asmuni, Hiishamuddin 59
Baarsma, Hilbrandt 547
Bai, Ruibin 67
Baptiste, Philippe 71
Barbulescu, Laura V. 259
Bautista Valhondo, Joaquín 550
Bellenguez-Morineau, Odile 608
Beyrouthy, Camille 553
Bilgintürk, Zehra 80
Billaut, Jean-Charles 498
Bist, Vineet 625
Blazewicz, Jacek 455
Bontoux, Boris 557
Boone, Anton 611
Bora, Selim 316
Boudhar, Mourad 88
Bourreau, Eric 557
Bozejko, Wojciech 96
Brauner, Nadia 581
Bräysy, Olli 136
Brucker, Peter 15
Bruno, Giuseppe 104
Burke, Edmund K. 59, 67, 418, 553, 595
Cesta, Amedeo 23
Chan, Peter 601
Chaovalitwongse, Wanpracha Art 316
Cheng, Eddie 267
Chrétienne, Philippe 561
Christoph, Habla 112
Davenport, Andrew 120
Davidović, Tatjana 128
De Causmaecker, Patrick 192
De Reyck, Bert 584
de Werra, Dominique 42
Deblaere, Filip 564
Della Croce, Federico 71
Demange, Marc 42
Demeulemeester, Erik L. 564, 572, 575
Do, Hoang 625
Dorne, Raphael 344
Dridi, Najoua 184
Dullaert, Wout 136
Dupont de Dinechin, Benoit 144
Ekim, Tinaz 42
Elvikis, Donatas 152
Ersoy, Ersan 159
Facó, João Lauro Dorneles 167
Fagundez, Fabio Dias 167
Feillet, Dominique 557
Finke, Gerd 24, 581
Fowler, John W. 112, 301, 591
García-Sabater, Jose Pedro 175
García-Sabater, Julio Juan 175
Garibaldi, Jonathan M. 59
Gawiejnowicz, Stanislaw 567
Gel, Esma 591
Gendreau, Michel 67
Godard, Daniel 276
Gourgand, Michel 251
Grosso, Andrea 71
Gupta, Jatinder N.D. 538
Hadda, Hatem 184
Hajri-Gabouj, Sonia 184
Hamacher, Horst W. 152
Hanzálek, Zdeněk 463
Hao, Jin-Kao 578
Haouari, Mohamed 217
Hasenfuss, Alexander 200
Haspeslagh, Stefann 192
Hecker, Klaus H. 200
Hemig, Claas 209
Hendel, Yann 570
Herbots, Jade 572
Herroelen, Willy S. 447, 572, 575, 611
Hiroux, Michael 601
Hmida, Abir Ben 217
Huguet, Marie-José 217
Hurink, Johann L. 547, 605
Improta, Gennaro 104
Jampani, Jagadish 225
Jansen, Pierre 547
Jarboui, Bassem 234
Kalagnanam, Jayant 120
Kalsh, Marcel T. 152
Karaesmen, Itir 242
Keha, Ahmet B. 301
Kemmoé Tchomté, Sylverin 251
Kendall, Graham 67, 595

Keskinocak, Pinar 242
Koksalan, Murat 591
Kononov, Alexander 567
Korhonen, Pekka 591
Kramer, Laurence A. 259
Kruk, Serge 267
Kubiak, Wieslaw 570
Kyngäs, Jari 386
Laborie, Philippe 276
Lacomme, Philippe 285
Lambrecht, Olivier 575
Lancia, Giuseppe 293
Landa-Silva, Dario 553
Larabi, Mohand 285
Lau, Hoong Chuin 328
Laub, Jeffrey 301
Laurent, Benoît 578
Lazarev, Alexander A. 309
Lehoux-Lebacque, Vassilissa 581
Lei, Lei 316
Leong, Thin Yin 328
Lesaint, David 344
Leung, Joseph Y.-T. 514
Leus, Roel 572, 584
Liberti, Leo 128
Limtanyakul, Kamol 336
Liret, Anne 344
Lopez, Pierre 217
Loucks, Wayne M. 368
Lucarelli, Giorgio 353
Maculan, Nelson 128
Maenhout, Broos 588
Marquis, Jon 591
Mason, Scott J. 225
McCollum, Barry 59, 553, 595
McMullan, Paul 553, 595
Melo, Rafael A. 431
Michel, Sophie 361
Milis, Ioannis 353
Miralles, Cristóbal 175
Mladenović, Nenad 128
Morton, Andrew 368
Mouloua, Zerouk 410
Müller, Tomáš 598
Munier-Kordon, Alix 377
Murray, Keith 598
Mönch, Lars 112
Naudin, Édith 601
Néron, Emmanuel 608
Nurmi, Kimmo 386
Oğuz, Ceyda 80
Özcan, Ender 159, 394
Pappis, Costas 402
Parkes, Andrew John 553
Paschos, Vangelis Th. 353
Paulus, Jacob Jan 605
Pereira Gude, Jordi 550
Pesch, Erwin 455
Pessan, Cédric 608
Petrovic, Sanja 506
Pfund, Michele E. 112
Pinedo, Michael 514
Pochet, Yves 621
Portmann, Marie-Claude 410
Qu, Rong 418
Quilliot, Alain 251
Raa, Birger 426
Rachaniotis, Nikos 402
Rapine, Christophe 581
Rebaï, Abdelwaheb 234
Rebaine, Djamal 377
Ribeiro, Celso C. 431
Righter, Rhonda 27
Rinaldi, Franca 293
Ruiz, Rubén 439
Sadykov, Ruslan 523
Salehipour, Amir 136
Salhi, Abdellah 506
Salman, Sibel 80
Schatteman, Damien 611
Schwiegelshohn, Uwe 336
Schwindt, Christoph 490
Serafini, Paolo 293
Sgalambro, Antonino 104
Sharma, Navin 625
Smith, Stephen F. 259
Soukhal, Ameur 498
Sourd, Francis 561
Speranza, Maria Grazia 51
Spieksma, Frits C.R. 33
Steiner, George 447
Sterna, Malgorzata 455
Šůcha, Přemysl 463
Szczerbicka, Helena 471
Tanaka, Shunji 614
Tchernev, Nicolay 285

Tchikou, Hamza 88
Thomas, Michael 471
T'kindt, Vincent 71
Touati, Sid-Ahmed-Ali 480
Trautmann, Norbert 490
Tuong, Nguyen Huynh 498
Tuza, Zsolt 51
Urrutia, Sebastián 431
Uyar, A. Şima 159
Van de Vonder, Stijn 611
Van Hentenryck, Pascal 41
Van Peteghem, Vincent 618
Vanden Berghe, Greet 192
Vanderbeck, François 361
Vanhoucke, Mario 618
Vanhoucke, Mario 588
Vázquez Rodríguez, José Antonio 506
Voudouris, Chris 344
Wallenius, Jyrki 591
Wan, Guohua 514
Warichet, Francois 621
Weil, Georges 601
White, George M. 625
Winter, Emilie 523
Wodecki, Mieczyslaw 96
Yang, Wei 242
Zémmouri, Tahar 601
Zhang, Xiandong 538
Zhang, Rui 447
Zimmermann, Jürgen 209
Zinder, Yakov 531