mista

1st Multidisciplinary International
Conference on Scheduling
Theory and Applications

MULTIDISCIPLINARY SCHEDULING

Theory and Applications

Graham Kendall, Edmund Burke
Sanja Petrovic and Michel Gendreau

Multidisciplinary Scheduling: Theory and Applications

Multidisciplinary Scheduling: Theory and Applications

1st International Conference, MISTA '03
Nottingham, UK, 13-15 August 2003
Selected Papers

edited by

Graham Kendall
Edmund Burke
Sanja Petrovic
Michel Gendreau

Springer

Graham Kendall
Univ. of Nottingham
United Kingdom

Edmund K. Burke
Univ. of Nottingham
United Kingdom

Sanja Petrovic
Univ. of Nottingham
United Kingdom

Michel Gendreau
Université de Montréal
Canada

Library of Congress Cataloging-in-Publication Data
A C.I.P. Catalogue record for this book is available
from the Library of Congress.

ISBN 0-387-25266-5 e-ISBN 0-387-25267-3 Printed on acid-free paper.

Printed in the United States of America.

9 8 7 6 5 4 3 2 1 SPIN 11052258

springeronline.com

Table of Contents

Machine Scheduling

Bin Packing

Educational Timetabling

Sports Scheduling

Transport Scheduling

Preface

The First Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA) was held in Nottingham, UK on 13–15th August 2003. Over one hundred people attended the conference and 67 abstracts and papers (including four plenary papers) were presented. All of these presentations were chosen for the conference after being refereed by our international Programme Committee which consisted of 90 scheduling researchers from across 21 countries and from across a very broad disciplinary spectrum (see below). After the conference, we invited the authors of the 67 accepted presentations to submit a full paper for publication in this post conference volume of selected and revised papers. This volume contains the 19 papers that were successful in a second round of rigorous reviewing that was undertaken (once again) by our Programme Committee.

The main goal of the MISTA conference series is to act as an international forum for multidisciplinary scheduling research. As far as we are aware, there is no other conference which is specifically aimed at exploring the interdisciplinary interactions which are so important (in our opinion) to future progress in scheduling research. As such, MISTA aims to bring together researchers from across disciplinary boundaries and to promote an international multi-disciplinary research agenda. The first conference was particularly successful in bringing together researchers from many disciplines including operational research, mathematics, artificial intelligence, computer science, management, engineering, transport, business and industry.

MISTA was one of the outcomes of a highly successful interdisciplinary scheduling network grant (GR/N35205) which was funded by the UK's Engineering and Physical Sciences Research Council (EPSRC)—which is the largest of the seven UK research councils. The network was launched in May 2001 and was funded for a period of three years. It provided an interdisciplinary framework for academia and industrialists to meet, exchange ideas and develop a collaborative multi-disciplinary scheduling research agenda. The MISTA conference was the culmination of the network's dissemination activity and it enabled the network to reach out to an international audience. The aim is that the MISTA conference series will become an ongoing international legacy of the network's activity.

The International Society for Interdisciplinary Scheduling (ISIS) was another initiative which arose from the network. Indeed, this society represents the network's international continuation strategy. The goal is that the society will carry the network's activity forward–but from an international rather than national perspective. The society currently has a healthy and growing mem-

bership and is open to anybody with an interest in interdisciplinary scheduling. The Journal of Scheduling (published by Kluwer) represents the society's journal and the MISTA conference represents the society's main international event.

The first MISTA conference could not have taken place without the help and support of many people and organisations. We would, firstly, like to acknowledge the support of EPSRC, the London Mathematical Society, Sherwood Press Ltd, Kluwer Academic Publishers and the University of Nottingham who all supported the conference with help, advice and (most importantly) financial contributions. We are particularly grateful to our international Programme Committee who worked extremely hard over two separate rounds of reviewing to ensure that the standards of the conference, and of this volume, were of the highest quality. We are very grateful to our Local Organising Committee (see below) who spent a significant amount of time to make sure that the conference ran smoothly. Very special thanks go to Alison Payne, Eric Soubeiga and Dario Landa Silva who deserve a particular mention for their hard work which really was above and beyond the call of duty. Thanks should also go to everybody else in the Automated Scheduling, Optimisation and Planning research group at Nottingham who all pulled together to help with all the little jobs that needed carrying out during the conference organisation. Special thanks should go to our copy editor, Piers Maddox, who has done such a wonderful job of putting this volume together in such a professional and careful manner. We would also like to acknowledge the significant support that we received from Gary Folven and his staff at Kluwer which was so important in launching a brand new conference series. We would like to say a particularly big thank you to the International Advisory Committee for their past and ongoing work in bringing you this and future MISTA conferences. Finally, we would like to thank the authors, delegates and (in particular) our plenary speakers for making the conference the great success that it was.

The Second MISTA conference is due to take place in New York on 18–20th July 2005. We are looking forward to it and we hope to see you there.

Graham Kendall
Edmund Burke
Sanja Petrovic
Michel Gendreau

October 2004

MISTA Conference Series
International Advisory Committee

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

MISTA 2003 Programme Committee

Graham Kendall (co-chair)	The University of Nottingham, UK
Edmund Burke (co-chair)	The University of Nottingham, UK
Sanja Petrovic (co-chair)	The University of Nottingham, UK
Michel Gendreau (co-chair)	Université de Montréal, Canada
Uwe Aickelin	The University of Bradford, UK
Hesham Alfares	King Fahd University of Petroleum & Minerals, Saudi Arabia
Abdelhakim Artiba	Facultes Universitares Catholiques de Mons (CREGI - FUCAM), Belguim
Belarmino Adenso-Diaz	University of Oviedo, Spain
Philippe Baptise	IBM T. J. Watson Research Centre, USA
James Bean	Department of Industrial and Operations Engineering, University of Michigan, USA
Jacek Blazewicz	Institute of Computing Science, Poznan University of Technology, Poland
Joachim Breit	Saarland University, Germany
Peter Brucker	University of Osnabrueck, Germany
Xiaoqiang Cai	The Chinese University of Hong Kong, Hong Kong
Jacques Carlier	Compiègne cedex France
Edwin Cheng	The Hong Kong Polytechnic University, Hong Kong
Philippe Chretienne	Paris 6 University, France
Ed Coffman	Columbia University, USA
Peter Cowling	The University of Bradford, UK
Patrick De Causmaecker	KaHo St.-Lieven, Ghent, Belgium
Mauro Dell'Amico	University of Modena and Reggio Emilia, Italy
Erik Demeulemeester	Katholieke Universiteit Leuven, Belgium
Kath Dowsland	Gower Optimal Algorithms Ltd, UK
Andreas Drexl	University of Kiel, Germany
Moshe Dror	University of Arizona, USA
Maciej Drozdowski	Poznan University of Technology, Poland
Janet Efstathiou	University of Oxford, UK
Wilhelm Erben	FH Konstanz - University of Applied Sciences, Germany

Dror Feitelson	The Hebrew University, Israel
Gerd Finke	Laboratory LEIBNIZ-IMAG, Grenoble, France
Peter Fleming	University of Sheffield, UK
David Fogel	Natural Selection, USA
Dalibor Froncek	University of Minnesota, USA
Celia A. Glass	Department of Actuarial Sciences and Statistics, City University, UK
Fred Glover	Leeds School of Business, University of Colorado, USA
Bernard Grabot	Laboratoire Génie de Production - Equipe Production Automatisée, France
Alain Guinet	Industrial Engineering Department, INSA de Lyon, France
Jin-Kao Hao	University of Angers, France
Martin Henz	National University of Singapore, Singapore
Jeffrey Herrmann	University of Maryland, USA
Willy Herroelen	Department of Applied Economics, Katholieke Universiteit Leuven, Belgium
Han Hoogeveen	Utrecht University, The Netherlands
Toshihide Ibaraki	Kyoto University, Japan
Jeffrey Kingston	University of Sydney, Australia
Hiroshi Kise	Kyoto Institute of Technology, Japan
Wieslaw Kubiak	MUN, Canada
Raymond Kwan	University of Leeds, UK
Claude Le Pape	ILOG, France
Chung-Yee Lee	The Hong Kong University of Science and Technology, Hong Kong
Arne Løkketangen	Department of Informatics, Molde College, Norway
Dirk C. Mattfeld	University of Bremen, Germany
David Montana	BBN Technologies, USA
Martin Middendorf	University of Leipzig, Germany
Alix Munier	LIP6, University Paris 12, France
Alexander Nareyek	Carnegie Mellon University, USA
Klaus Neumann	University of Karlsruhe, Germany
Bryan A. Norman	University of Pittsburgh, USA
Wim Nuijten	ILOG, France
Ibrahim Osman	American University of Beirut, Lebanon
Costas P. Pappis	University of Piraeus, Greece
Erwin Pesch	University of Siegen, Germany

Dobrila Petrovic	Coventry University, UK
Michael Pinedo	New York University, USA
Chris Potts	University of Southampton, UK
Christian Prins	University of Technology, Troyes, France
Jean-Yves Potvin	Université de Montreal, Canada
Kirk Pruhs	University of Pittsburgh, USA
Vic J. Rayward-Smith	University of East Anglia, UK
Colin Reeves	Coventry University, UK
Celso C. Ribeiro	Catholic University of Rio de Janeiro, Brazil
Andrea Schaerf	University of Udine, Italy
Guenter Schmidt	Saarland University, Germany
Roman Slowinski	Poznan University of Technology, Poland
Stephen Smith	Carnegie Mellon University, USA
Vincent T'Kindt	University of Tours, France
Roberto Tadei	Politecnico di Torino, Italy
Jonathan Thompson	Cardiff University, UK
Michael Trick	Graduate School of Industrial Administration, Carnegie Mellon University, USA
Edward Tsang	University of Essex, UK
Denis Trystram	ID - IMAG, France
Steef van de Velde	Erasmus University, Netherlands
Greet Vanden Berghe	KaHo St.-Lieven, Ghent, Belgium
Stefan Voss	University of Hamburg, Germany
Jan Weglarz	Poznan University of Technology, Poland
Dominique de Werra	IMA, Faculté des Sciences de Base, Lausanne, Switzerland
George White	University of Ottawa, Canada
Darrell Whitley	Colorado State University, USA
Gerhard J. Woeginger	Faculty of Mathematical Sciences, University of Twente, The Netherlands
Yakov Zinder	University of Technology, Sydney, Australia
Qingfu Zhang	University of Essex, UK

MISTA 2003 Local Organizing Committee

Samad Ahmadi	De Montfort University, UK
Edmund Burke	University of Nottingham, UK
Diana French	University of Nottingham, UK
Jon Garibaldi	University of Nottingham, UK
Steven Gustafson	University of Nottingham, UK
Graham Kendall (chair)	University of Nottingham, UK
Natalio Krasnogor	University of Nottingham, UK
Dario Landa	University of Nottingham, UK
Djamila Ouelhadj	University of Nottingham, UK
Alison Payne	University of Nottingham, UK
Eric Soubeiga	University of Nottingham, UK
Sanja Petrovic	University of Nottingham, UK
Razali Bin Yaakob	University of Nottingham, UK

Fundamentals of Scheduling

IS SCHEDULING A SOLVED PROBLEM?

Stephen F. Smith
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213, USA
sfs@cs.cmu.edu

Abstract In recent years, scheduling research has had an increasing impact on practical problems, and a range of scheduling techniques have made their way into real-world application development. Constraint-based models now couple rich representational flexibility with highly scalable constraint management and search procedures. Similarly, mathematical programming tools are now capable of addressing problems of unprecedented scale, and meta-heuristics provide robust capabilities for schedule optimisation. With these mounting successes and advances, it might be tempting to conclude that the chief technical hurdles underlying the scheduling problem have been overcome. However, such a conclusion (at best) presumes a rather narrow and specialised interpretation of scheduling, and (at worst) ignores much of the process and broader context of scheduling in most practical environments. In this note, I argue against this conclusion and outline several outstanding challenges for scheduling research.

Keywords: scheduling.

1. STATE OF THE ART

More than once in the past couple of years, I have heard the opinion voiced that "Scheduling is a solved problem". In some sense, it is not difficult to understand this view. In recent years, the scheduling research community has made unprecedented advances in the development of techniques that enable better solutions to practical problems. In the case of AI-based scheduling research (the field I am most familiar with), there are now numerous examples of significant success stories. Constraint satisfaction search with dynamic backtracking has been used to successfully solve an avionics processor scheduling problem involving synchronisation of almost 20,000 activities under limited resources and complex temporal constraints (Boddy and Goldman, 1994). Program synthesis technology has been used to derive efficient constraint propagation code for large-scale deployment scheduling problems that has been

demonstrated to provide several orders of magnitude speed-up over current tools (Smith *et al.*, 1995). Genetic algorithm based scheduling techniques (Syswerda, 1991) have transitioned into commercial tools for optimising manufacturing production. Incremental, constraint-based scheduling techniques have been deployed for large-scale operations such as space shuttle ground processing (Zweben *et al.*, 1994) and day-to-day management of USAF airlift assets (Smith *et al.*, 2004a).

These examples of application successes reflect well on the effectiveness and relevance of underlying research in the field of scheduling. However, to extrapolate from such examples to the conclusion that the chief technical hurdles underlying the scheduling problem have now been overcome is a considerable leap. The scheduling research community has become a victim of its own success.

Summarising the current state of the art, we can indeed identify several technological strengths:

- *Scalability.* Current scheduling techniques are capable of solving large problems (i.e. tens of thousands of activities, hundreds of resources) in reasonable time frames.
- *Modelling flexibility.* Current techniques are capable of generating schedules under broad and diverse sets of temporal and resource capacity constraints.
- *Optimisation.* Research in applying various global, local and meta-heuristic based search frameworks to scheduling problems has produced a number of general approaches to schedule optimisation, and increasing integration of AI-based search techniques with mathematical programming tools (e.g. linear, mixed-integer constraint solvers) is yielding quite powerful optimisation capabilities.

Taken together, there is a fairly transferrable set of techniques and models for efficiently generating high quality schedules under a range of constraints and objectives.

On the other hand, claims that these technological strengths demonstrate that the scheduling problem is solved, and hence research funds and activity would be better focused elsewhere, must be considered more carefully. At best, these claims presume a narrow (perhaps classical) definition of scheduling as a static, well-defined optimisation task (a sort of puzzle solving activity). But, even under this restricted view of scheduling, one can argue that the conclusion is debatable. Despite the strengths of current techniques, the problems being addressed by current scheduling technologies are generally NP hard and solved only approximately; there is considerable room for improvement in techniques for accommodating different classes of constraints and for optimising under

different sets of objective criteria. However, at a broader level, scheduling is rarely a static, well-defined generative task in practice. It is more typically an ongoing, iterative process, situated in a broader planning/problem solving context, and more often than not involving an unpredictable and uncertain executing environment. Each of these additional aspects raises important and fundamental questions for scheduling research. The scheduling problem is far from solved.

2. RESEARCH CHALLENGES

Taking the broader view of scheduling just summarised, many important research challenges can be identified. Several are outlined in the sections below.

2.1 Generating Schedules under Complex Constraints, Objectives and Preferences

Though scheduling research has produced a substantial set of reusable tools and techniques, the generation of high-quality solutions to practical scheduling problems remains a custom art and is still confounded by issues of scale and complexity. More often than not, it is necessary to incorporate specialised heuristic assumptions, to treat selected constraints and objectives in an ad hoc manner, and more generally to take advantage of problem-specific solution engineering to obtain a result that meets a given application's requirements. There continues to be great need for research into techniques that operate with more realistic problem assumptions.

One broad area where prior research has tended to simplify problem formulation is in the treatment of scheduling objectives and preferences. Mainstream scheduling research has focused predominately on optimisation of selected, simple objective criteria such as minimising makespan or minimising tardiness. These objectives provide concise problem formulations but often bear little relationship to the requirements of practical domains. For example, most make-to-order manufacturing organisations strive to set reasonable due dates and avoid late deliveries; a scheduling objective such as minimising tardiness does not match this requirement. In many problems, there are multiple, conflicting objectives that must be taken into account. In others, there are complex sets of so-called "soft" constraints that should be satisfied if possible but do not necessarily have to be, and the problem is most naturally formulated as one of optimising the overall level of satisfaction of these preferences. In still other domains, the scheduling objective is tied to the expected *output* of the process rather than its efficiency, with the goal being to optimise the quality (or utility) of the tasks that can be executed within known deadline constraints. Recent work in such areas as multicriteria scheduling (Della Croce *et al.*, 2002; T'kindt and Billaut, 2002; Landa Silva and Burke, 2004), scheduling with

complex preferences (Burke and Petrovic, 2002) and scheduling to maximise process quality (Ajili and El Sakkout, 2003; Schwarzfischer, 2003; Wang and Smith, 2004), has made some progress in generating schedules that account for more realistic objective criteria, but there is considerable room for further research here.

A second continuing challenge is the design of effective heuristic procedures for generating high quality solutions to practical problems. There has been a large body of research into the design of scheduling rules and heuristics for various classes of scheduling problems (Morton and Pentico, 1993). Although such heuristics can be effective in specific circumstances, they are not infallible and their myopic nature can often give rise to suboptimal decisions. At the other extreme, meta-heuristic search techniques (Voss, 2001) provide a general heuristic basis for generating high quality solutions in many domains, but often require extended execution time frames to be effective.

One approach to overcoming the fallibility of scheduling heuristics is to exploit them within a larger search process. Systematic search techniques such as limited discrepancy search (Harvey and Ginsberg, 1995) and depth-bounded discrepancy search (Walsh, 1997) take this perspective; each starts from the assumption that one has a good heuristic, and progressively explores solutions that deviate in more and more decisions from the choices specified by the heuristic. A similarly motivated idea is to use a good heuristic to bias a non-deterministic choice rule and embed this randomised solution generator within an iterative sampling search process (Bresina, 1996; Oddi and Smith, 1997; Cicirello and Smith, 2002). In this case, the search is effectively broadened to cover the "neighbourhood" of the trajectory that would be defined by deterministically following the heuristic. Both of these approaches to using a heuristic to direct a broader search process been effectively applied to complex scheduling problems.

A second approach to overcoming the limitations of any one scheduling heuristic is to attempt to combine the use of several. It is rarely the case that a heuristic can be found that dominates all others in a particular domain. More frequently, different heuristics tend to perform better or worse on different problem instances. Following this observation, a number of recent approaches have begun to explore techniques that take advantage of several heuristics (or heuristic problem solving procedures) in solving a given instance of a scheduling problem. In some approaches (Talukdar *et al.*, 1998; Gomes and Selman, 2001) different heuristic search procedures are executed in parallel, with the possibility of sharing and building on intermediate results. In other work, the development of adaptive scheduling procedures is considered, which utilise some form of online learning procedure to determine which heuristic or heuristic procedure is best suited to solve each specific problem instance (Hartmann, 2002; Burke *et al.*, 2003; Cicirello and Smith, 2004b). Work in the direction of

combining multiple scheduling heuristics and procedures has produced some interesting and promising results. At the same time, there are still significant challenges in extending and scaling these ideas to meet the requirements of practical domains.

One important general direction for research into more effective schedule generation procedures is to explore integration of approximate and exact methods, and other cross-fertilisation of techniques that have emerged in different disciplines. Growing research activity in the area of combining constraint logic programming with classical optimisation (McAloon and Tretkoff, 1996; Hooker, 2000; Regin and Rueher, 2004), for example, has shown the potential for significant advances in solving complex and large-scale combinatorial problems, and this work is starting to find application in scheduling domains (Baptiste *et al.*, 2001; Hooker, 2004). Another important direction for future research is more principled analysis of scheduling search procedures. Recent work in this direction (Watson *et al.*, 1999; Watson, 2003) has produced results that show the inadequacy of using randomly generated problems as a basis for evaluating real-world algorithm performance and the importance of problem structure on algorithm design. Better understanding of the behaviour of search algorithms in scheduling search spaces should ultimately lead to development of more effective scheduling procedures.

2.2 Managing Change

If the goal of scheduling is to orchestrate an optimised behaviour of some resource-limited system or organisation over time, then the value of a schedule will be a function of its continuing relevance to the current environmental state. One can categorise scheduling environments along a continuum ranging from highly predictable and stable to highly uncertain and dynamic. Current techniques are best suited for applications that fall toward the predictable end of the spectrum, where optimised schedules can be computed in advance and have a reasonable chance of being executable. Many spacecraft mission planning and scheduling problems have this character. Although things can certainly go wrong (and do), predictive models of constraints are generally pretty accurate, and the time and cost put into obtaining the most optimised schedule possible is worth it.[1] Unfortunately, though, most practical applications tend to fall more toward the other end of the continuum, where advance schedules can have a very limited lifetime and scheduling is really an ongoing process of responding to unexpected and evolving circumstances. In such environments, insurance of robust response is generally the first concern.

[1] An extreme example was the most recent Jupiter flyby, where it is estimated that somewhere on the order of 100,000 person hours went into construction of the 1–2 week observing schedule (Biefeld, 1995).

Managing change to schedules in such dynamic environments remains a significant challenge. For any sort of advance schedule to be of ongoing value, the scheduler (or re-scheduler) must be capable of keeping pace with execution. But even supposing this is not a problem, it is typically not sufficient to simply re-compute from scratch with a suitably revised starting state. When multiple executing agents are involved (as is the case in most scheduling applications), wheels are set in motion as execution unfolds and there is a real cost to repeatedly changing previously communicated plans. Explicit attention must be given to preserving stability in the schedule over time and localising change to the extent possible. While there has been some work in this direction over the past several years (Smith, 1994; Zweben *et al.*, 1994; El Sakkout and Wallace, 2000; Montana *et al.*, 1998; Bierwirth and Mattfeld, 1999; Zhou and Smith, 2002; Kramer and Smith 2003, 2004; Hall and Posner, 2004), there is still little understanding of strategies and techniques for explicitly trading off optimisation and solution continuity objectives.

An alternative approach to managing execution in dynamic environments is to build schedules that retain flexibility and hedge against uncertainty. Work to date has focused principally on scheduling techniques that retain various forms of temporal flexibility (e.g. Smith and Cheng, 1993; Cesta *et al.*, 1998; Artigues *et al.*, 04; Leus and Herroelen, 2004; Policella *et al.*, 2004a) and on transformation of such schedules into a form that enables efficient execution (Muscettola *et al.*, 1998, Wallace and Freuder, 2000). A similar concept of producing solutions that promote bounded, localised recovery from execution failures is proposed in Ginsberg *et al.* (1998) and also explored in Branke and Mattfeld (2002) and Hebrard *et al.* (2004). However, with few exceptions these approaches take a strict constraint satisfaction perspective, and exploit flexibility only as defined by current time and capacity constraints. Only recently (e.g. Aloulou and Portmann, 2003; Policella *et al.*, 2004b) has any work considered the problem of generating flexible schedules in the presence of objective criteria. Likewise, strategies for intelligently inserting flexibility into the schedule based on information or knowledge about various sources of uncertainty (e.g. mean time to failure, operation yield rates) have received only limited attention (e.g. Mehta and Uzsoy, 1998; Davenport *et al.*, 2001) and remain largely unexplored. A somewhat related idea is to use uncertainty information as a basis for developing contingent schedules. This approach is taken in Drummond *et al.* (1994) to deal with activity duration uncertainty. Other recent work (McKay *et al.*, 2000; Black *et al.*, 2004) has focused on the development of context-sensitive scheduling rules, which adjust job priorities in the aftermath of unexpected events to minimise deleterious consequences.

2.3 Self-Scheduling Systems

A third approach to managing execution in dynamic environments that has gained increasing attention in recent years involves the development of so-called self-scheduling systems, where (in the extreme) schedules are not computed in advance but instead scheduling decisions are made only as needed to keep execution going. Such systems are composed of a collection of interacting decision-making agents, each responsible for brokering the services of one or more resources, managing the flow of particular processes, etc. Agents coordinate locally to make various routing and resource assignment decisions and global behaviour is an emergent consequence of these local interactions.

Such approaches are attractive because they offer robustness and simplicity, and there have been a few interesting successes (Morley and Schelberg, 1992). At the same time, these approaches make no guarantees with respect to global performance, and very simple systems have been shown to have tendencies toward chaotic behaviour (Beaumariage and Kempf, 1995). Some recent work has approached this coordination problem as an adaptive process and has leveraged naturally-inspired models of adaptive behaviour to achieve coherent global behaviour in specific manufacturing control contexts (Parunak *et al.*, 1998; Campos *et al.*, 2000; Cicirello and Smith 2004a). But speaking generally, the problem of obtaining good global performance via local interaction protocols and strategies remains a significant and ill-understood challenge.

Self-scheduling approaches do not preclude the computation and use of advance schedules, and indeed their introduction may offer an alternative approach to overcoming the above-mentioned tendencies toward sub-optimal global performance. Distributed, multi-agent scheduling models are also important in domains where problem characteristics (e.g. geographical separation, authority, security) prohibit the development of centralised solutions. A number of agent-based approaches, employing a variety of decomposition assumptions and (typically market-based) interaction protocols, have been investigated over the past several years (Malone *et al.*, 1988; Ow *et al.*, 1988; Sycara *et al.*, 1991; Lin and Solberg, 1992; Liu, 1996; Montana *et al.*, 2000; Wellman *et al.*, 2001; Goldberg *et al.*, 2003). More recently, protocols and mechanisms for incremental, time-bounded optimisation of resource assignments (Mailler *et al.*, 2003; Wagner *et al.*, 2004) and for self-improving self-scheduling systems (Oh and Smith, 2004) have begun to be explored. However, the question of how to most effectively coordinate resource usage across multiple distributed processes is still very much open.

2.4 Integrating Planning and Scheduling

Though scheduling research has historically assumed that the set of activities requiring resources can be specified in advance, a second common charac-

teristic of many practical applications is that planning—the problem of determining which activities to perform, and scheduling—the problem of allocating resources over time to these activities, are not cleanly separable. Different planning options may imply different resource requirements, in which case the utility of different planning choices will depend fundamentally on the current availability of resources. Similarly, the allocation of resources to a given activity may require a derivative set of enabling support activities (e.g. positioning, reconfiguration) in which case the specification and evaluation of different scheduling decisions involves context-dependent generation of activities. Classical "waterfall" approaches to decision integration, where planning and scheduling are performed in sequential lockstep, lead to lengthy inefficient problem solving cycles in these sorts of problems.

The design of more tightly integrated planning and scheduling processes is another important problem that requires research. One approach is to represent and solve the full problem in a single integrated search space. A survey of such approaches can be found in Smith *et al.* (2000). However, use of a common solver typically presents a very difficult representational challenge. It has also recently been shown that the use of separable planning and scheduling components can offer computational leverage over a comparable integrated model, due to the ability to exploit specialised solvers (Srivastava *et al.*, 2001). In resource-driven applications, where planning is localised to individual jobs, it is sometimes possible to incorporate planning conveniently as a subsidiary process to scheduling (Muscettola *et al.*, 1992; Sadeh *et al.*, 1998; Chien *et al.*, 1999; Smith *et al.*, 2003; Smith and Zimmerman, 2004). For more strategy-oriented applications, though, where inter-dependencies between activities in the plan are less structured and more goal dependent, it is necessary to develop models for tighter and more flexible interleaving of planning and scheduling decisions. One such model, based on the concept of customising the plan to best exploit available resources, is given in Myers *et al.* (2001).

2.5 Requirements Analysis

Despite the ultimate objective of producing a schedule that satisfies domain constraints and optimises overall performance, scheduling in most practical domains is concerned with solving a problem of much larger scope, which additionally involves the specification, negotiation and refinement of input requirements and system capabilities. This larger process is concerned most basically with getting the constraints right: determining the mix of requirements and resource capacity that leads to most effective overall system performance.

It is unreasonable to expect to fully automate this requirements analysis process. The search space is unwieldy and ill-structured, and human expertise is needed to effectively direct the search process. At the same time, problem

scale generally demands substantial automation. The research challenge is to flexibly inject users into the scheduling process, without requiring the user to understand the system's internal model. In other words, the system must bear the burden of translating to and from user interpretable representations, conveying results in a form that facilitates comprehension and conveys critical tradeoffs, and accepting user guidance on how to next manipulate the system model.

Work to date toward the development of mixed-initiative scheduling systems has only taken initial steps. One line of research has focused on understanding human aspects of planning and scheduling, examining the planning and scheduling processes carried out in various organisations and analysing the performance of human schedulers in this context (McKay *et al.*, 1995; MacCarthy and Wilson, 2001). This work has provided some insight into the roles that humans and machines should assume to maximise respective strengths, and in some cases guidance into the design of more effective practical scheduling techniques. But there are still fairly significant gaps in understanding how to integrate human and machine scheduling processes.

From the technology side, there has been some initial work in developing interactive systems that support user-driven scheduling. In Smith *et al.* (1996), Becker and Smith (2000), and Smith *et al.* (2004a) parametrisable search procedures are used in conjunction with graphical displays to implement a "spreadsheet" like framework for generating and evaluating alternative constraint relaxation options. Ferguson and Allen (1998) alternatively exploit a speech interface, along with techniques for dialogue and context management, to support collaborative specification of transportation schedules. An interactive 3D visualisation of relaxed problem spaces is proposed in Derthick and Smith (2004) as a means for early detection and response to capacity shortfalls caused by conflicting problem requirements. In Smith *et al.* (2003, 2004b), some preliminary steps are taken toward exploiting a scheduling domain ontology as a basis for generating user-comprehensible explanations of detected constraint conflicts. But in general there has been very little investigation to date into techniques for conveying critical decision tradeoffs, for explaining system decisions and for understanding the impact of solution change.

2.6 E-Commerce Operations

Finally, I mention the emerging application area of Electronic Commerce as a rich source for target problems and an interesting focal point for scheduling research. Current electronic marketplaces provide support for matching buyers to suppliers (and to a lesser extent for subsequent procurement and order processing). However, once the connection is made, buyers and suppliers leave the eMarketplace and interact directly to carry out the mechanics of order ful-

filment. In the future, one can easily envision expansion of the capabilities of eMarketplaces to encompass coordination and management of subscriber supply chains. Such E-Commerce operations will include available-to-promise projection and due date setting, real-time order status tracking, determination of options for filling demands (optimised according to specified criteria such as cost, lead-time, etc.) and order integration across multiple manufacturers.

All of these capabilities rely rather fundamentally on a flexible underlying scheduling infrastructure, and taken collectively they provide a strong forcing function for many of the research challenges mentioned earlier. Scheduling techniques that properly account for uncertainty, enable controlled solution change, and support efficient negotiation and refinement of constraints, are crucial prerequisites, and the need to operate in the context of multiple self-interested agents is a given. The advent of E-Commerce operations also raises some potentially unique challenges in scheduling system design and configuration, implying the transition of scheduling and supply chain coordination technologies from heavyweight back-end systems into lightweight and mobile services.

3. CONCLUSIONS

The field of scheduling has had considerable success in recent years in developing and transitioning techniques that are enabling better solutions to practical scheduling applications. Given this success, it might be tempting to conclude that major technical and scientific obstacles have now been cleared. In this brief note, I have argued against this notion and highlighted several outstanding challenges for scheduling research. There is plenty that remains to be done.

Acknowledgments

Thanks are due to Larry Kramer and David Hildum for useful comments on an earlier draft of this paper. This work was supported in part by the Department of Defence Advanced Research Projects Agency and the U.S. Air Force Research Laboratory, Rome under contracts F30602-97-2-0666 and F30602-00-2-0503, by the National Science Foundation under contract 9900298 and by the CMU Robotics Institute.

References

Ajili, F. and El Sakkout, H. (2003) A probe-based algorithm for piecewise linear optimization in scheduling. *Annals of Operations Research*, **118**:35–48.

Aloulou, M. A. and Portmann, M.-C. (2003) An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. In *Proceedings of the 1st MultiDisciplinary International Conference on Scheduling Theory and Applications*, Nottingham UK, pp. 337–362.

Artigues, C., Billaut, J.-C. and Esswein, C. (2004) Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, to appear.

Baptiste, P., LePape, C. and Nuijten, W. (2001) *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Kluwer, Dordrecht.

Beaumariage, T. and Kempf, K. (1995). Attractors in manufacturing systems with chaotic tendencies. *INFORMS-95 Presentation*, New Orleans.

Becker, M. and Smith, S. F. (2000). Mixed-initiative resource management: the amc barrel allocator. In *Proceedings of the 5th International Conference on AI Planning and Scheduling* (Breckenridge, CO), AAAI Press, Menlo Park, CA, pp. 32–41.

Bierwirth, C. and Mattfeld, D. C. (1999). Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, **7**:1–17.

Biefeld, E. (1995) Private communication.

Black, G. W., McKay, K. N. and Messimer, S. L. (2004) Predictive, stochastic and dynamic extensions to aversion dynamics scheduling. *Journal of Scheduling*, **7**:277–292.

Boddy, M. and Goldman, R. P. (1994) Empirical results on scheduling and dynamic backtracking. In *Proceedings of the 3rd International Symposium on Artificial Intelligence, Robotics and Automation for Space,* Pasadena CA, pp. 431–434.

Branke, J. and Mattfeld, D. C. (2002), Anticipatory scheduling for dynamic job shops. *AIPS-02 Workshop on On-line Planning and Scheduling,* Toulouse, France, G. Verfaillie (Ed.), pp. 3–10.

Bresina, J. (1996) Heuristic-biased stochastic sampling. In *Proceedings of the 13th National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, pp. 271–278.

Burke, E. K. and Petrovic, S. (2002) Recent research trends in automated timetabling. *European Journal of Operational Research*, **140**:266–280.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S. (2003) Hyper-heuristics: an emerging direction in modern search technology. In *Handbook of Meta-Heuristics*, F. Glover and G. Kochenberger (Eds.) Kluwer, Dordrecht, pp. 457–474.

Campos, M. , Bonabeau, E., Theraulaz, G. and Deneubourg, J. (2000) Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8:83–96.

Cesta, A., Oddi, A. and Smith, S. F. (1998) Profile-based algorithms for solving multi-capacitated metric scheduling problems. In *Proceedings of the 4th International Conference on AI Planning and Scheduling*, Pittsburgh PA, pp. 214–223.

Chien, S., Rabideau, G., Willis, J. and Mann, T. (1999) Automated planning and scheduling of shuttle payload operations. *Artificial Intelligence*, **114**:239–255.

Cicirello, V. and Smith, S. F. (2002) Amplification of search performance through randomization of heuristics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming* (Ithaca, NY), Lecture Notes in Computer Science, Vol. 2470, pp. 124–138.

Cicirello, V. and Smith, S. F. (2004a) Wasp-based Agents for Distributed Factory Coordination. *Journal of Autonomous Agents and Multi-Agent Systems*, **8**:237–266.

Cicirello, V. and Smith, S. F. (2004b) Heuristic selection for stochastic search optimization: modeling solution quality by extreme value theory. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, Toronto, CA.

Davenport, A. J., Gefflot, C. and Beck, J. C. (2001) Slack-based techniques for building robust schedules. In *Proceedings of the 6th European Conference on Planning* (Toledo, Spain), pp. 181–192.

Della Croce, F., Tsoukias, A. and Moraitis, P. (2002) Why is it difficult to make decisions under multiple criteria? In *Working Notes AIPS-02 Workshop on Planning and Scheduling with Multiple Criteria* (Toulouse, France), pp. 41–45.

Derthick, M. and Smith, S. F. (2004) An interactive 3D visualization for requirements analysis. *Robotics Institute Technical Report CMU-RI-TR-04-65*, Carnegie Mellon University, Pittsburgh, PA.

Drummond, M., Bresina, J. and Swanson, K. (1994) Just-in-case scheduling. In *Proceedings of the 12th National Conference on Artificial Intelligence* (Seattle, WA), pp.1098–1104.

El Sakkout, H. and Wallace, M. (2000) Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, **5**:359–388.

Ferguson, G. and Allen, J. (1998) TRIPS: An integrated intelligent problem-solving assistant. *Proceedings of the 15th National Conference on Artificial Intelligence* (Madison, WI) pp. 567–572.

Ginsberg, M., Parkes, A. and Roy, A. (1998) Supermodels and robustness. In *Proceedings of the 15th National Conference on Artificial Intelligence* (Madison, WI), pp. 334–339.

Goldberg, D., Cicirello, V., Dias, M. B., Simmons, R., Smith, S. F. and Stentz, T. (2003) Market-based multi-robot planning in a distributed layered architecture. In *Proceedings of the 2nd International Workshop on Multi-Robot Systems* (Washington, DC), Kluwer, Dordrecht, pp. 27–38.

Gomes, C. P. and Selman, B. (2001) Algorithm portfolios. *Artificial Intelligence*, **126**:43–62.

Hall, N. G. and Posner, M. E. (2004) Sensitivity analysis for scheduling problems. *Journal of Scheduling*, **7**:49–83.

Hartmann, S. (2002) A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, **49**:433–448.

Harvey, W. and Ginsberg, M. (1995) Limited discrepancy search. *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (Montreal, QC), Morgan Kaufmann, pp. 607–615.

Hebrard, E., Hnich, B. and Walsh, T. (2004) Robust solutions for constraint satisfaction and optimization. In *Proceedings of the European Conference on Artificial Intelligence*.

Hooker, J. N. (2000) *Logic-Based Methods for Optimization*, Wiley, New York.

Hooker, J. N. (2004) A hybrid method for planning and scheduling. In *Principles and Practice of Constraint Programming (CP 2004)* (Toronto, ON), Lecture Notes in Computer Science, Vol. 3258, M. Wallace (Ed.), pp. 305–316.

Kramer, L. A. and Smith, S. F. (2003) Max-flexibility: a retraction heuristic for over-subscribed scheduling problems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)* (Acapulco, Mexico), pp. 1218–1223.

Kramer, L. A. and Smith, S. F. (2004) Task Swapping for Schedule Improvement: A Broader Analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling* (Whistler, BC), pp. 235–243.

Landa Silva, J. D. and Burke, E. K. (2004) A tutorial on multiobjective metaheuristics for scheduling and timetabling. In *Multiple Objective Meta-Heuristics*, X. Gandibleux, M. Sevaux, K. Sorensen and V. T'Kindt (Eds.), Lecture Notes in Economics and Mathematical Systems, Springer, Berlin.

Leus, R. and Herroelen, W. (2004) Stability and resource allocation in project planning. *IIE Transactions*, **36**:667–682.

Lin, G. and Solberg, J. (1992) Integrated shop floor control using autonomous agents. *IIE Transactions*, **24**:57–71.

Liu, J. S. (1996). Coordination of multiple agents in distributed manufacturing scheduling. *Ph.D. Thesis*, The Robotics Institute, Carnegie Mellon University.

MacCarthy, B. L. and Wilson, J. R. (Eds.) (2001) *Human Performance in Planning and Scheduling*, Taylor and Francis, New York.

Mailler, R., Lesser, V. and Horling, B. (2003) Cooperative negotiation for soft real-time distributed resource allocation. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 576–583.

Malone, T. W., Fikes, R. E., Grant, K. R. and Howard, M. T. (1988) Enterprise: a market-like task scheduler for distributed computing environments. In *The Ecology of Computation*, B. A. Huberman (Ed.), Elsevier, Amsterdam.

McAloon, K. and Tretkoff, C. (1996) *Optimization and Computational Logic*, Wiley, New York.

McKay, K. N., Safayeni, F. R. and Buzacott, J. A. (1995) Schedulers and planners: what and how can we learn from them? In *Intelligent Scheduling Systems*, D. E. Brown and W. T. Scherer (Eds.), Kluwer, Dordrecht, pp. 41–62.

McKay, K. N., Morton, T. E., Ramnath, P. and Wang, J. (2000) "Aversion dynamics" scheduling when the system changes. *Journal of Scheduling*, **3**:71–88.

Mehta, S. and Uzsoy, R. (1998). Predictable scheduling of a job-shop subject to breakdowns, *IEEE Transactions on Robotics and Automation,* **14**:365–378.

Montana, D., Brin, M., Moore, S. and Bidwell, G. (1998) Genetic algorithms for complex, real-time scheduling. In *Proceedings of the 1998 IEEE Conference on Man, Systems and Cybernetics.*

Montana, D., Herrero, J., Vidaver, G. and Bidwell, G. (2000) A multiagent society for military transportation scheduling. *Journal of Scheduling*, **3**:225–246.

Morley, D. and Schelberg, C. (1992) An analysis of a plant specific dynamic scheduler. *Final Report, NSF Workshop on Intelligent Dynamic Scheduling for Manufacturing Systems*, L. Interrante and S. F. Smith (Eds.).

Morton, T. E. and Pentico, D. (1993) *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*, Wiley, New York.

Muscettola, N., Smith, S. F., Cesta, A. and D'Aloisi, D. (1992) Coordinating space telescope operations in an integrated planning and scheduling framework. *IEEE Control Systems*, **12**:28–37.

Muscettola, N., Morris, P. and Tsamardinos, I. (1998) Reformulating temporal plans for efficient execution. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 444–452.

Myers, K. L., Smith, S. F., Hildum, D., Jarvis, P. and de Lacaze, R. (2001) Integrating planning and scheduling through adaptation of resource intensity estimates. In *Proceedings of the 6th European Conference on Planning* (Toledo, Spain), pp. 133–144.

Oddi, A. and Smith, S. F. (1997) Stochastic procedures for generating feasible schedules. In *Proceedings of the 14th National Conference on Artificial Intelligence* (Providence, RI), pp. 308–314.

Oh, J. and Smith, S. F. (2004) Learning user preferences in distributed calendar scheduling. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling* (Pittsburgh, PA).

Ow, P. S., Smith, S. F. and Howie, R. (1988) A cooperative scheduling system. In *Expert Systems and Intelligent Manufacturing*, M. Oliff (Ed.), Elsevier, Amsterdam.

Parunak, V., Baker, A. and Clark, S. (1998) The AARIA agent architecture: from manufacturing requirements to agent-based system design. In *Proceedings of the ICRA'98 Workshop on Agent-based Manufacturing* (Minneapolis, MN), pp. 1–17.

Policella, N., Smith, S. F., Cesta, A. and Oddi, A. (2004a) Generating robust schedules through temporal flexibility. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling* (Whistler, BC), pp. 209–218.

Policella, N., Oddi, A., Smith, S. F. and Cesta, A. (2004b) Generating robust partial order schedules. In *Principles and Practice of Constraint Programming, 10th International Conference*

(CP 2004) (Toronto, ON), M. Wallace (Ed.), Lecture Notes in Computer Science, Vol. 3258, Springer, Berlin.

Regin, J.-C. and Rueher, M. (Eds.) (2004) *Proceedings CP-AI-OR'04: International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Nice, France).

Sadeh, N., Hildum, D. W., LaLiberty, T. J., McNulty, J., Kjenstad, D. and Tseng, A. (1998) A blackboard architecture for integrating process planning and production scheduling. *Concurrent Engineering: Research and Applications*, **6**:88–100.

Schwarzfischer, T. (2003) Quality and utility—towards a generalization of deadline and anytime scheduling. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling* (Trento, Italy), pp. 277–286.

Smith, D. E., Frank, J. and Jonsson, A. K. (2000) Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, **15**:61–94.

Smith, D. R., Parra, E. and Westfold, S. (1995). Synthesis of high performance transportation schedulers. *Technical Report* KES.U.95.1, Kestrel Institute.

Smith, S. F. and Cheng, C. (1993) Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence* (Washington, DC), pp. 139–144.

Smith, S. F. (1994) OPIS: An Architecture and Methodology for Reactive Scheduling. In *Intelligent Scheduling*, Chapter 2, M. Zweben and M. Fox (Eds.), Morgan Kaufmann, San Mateo, CA.

Smith, S. F., Lassila, O. and Becker, M. (1996) Configurable systems for mixed-initiative planning and scheduling. In *Advanced Planning Technology*, A. Tate (Ed.), AAAI Press, Menlo Park, CA. ISBN 0-929280-98-9.

Smith, S. F., Becker, M. A. and Kramer, L. A. (2004a) Continuous management of airlift and tanker resources: a constraint-based approach. *Journal of Mathematical and Computer Modeling*, Special Issue on Defence Transportation: Algorithms, Models and Applications for the 21st Century, **39**:581–598.

Smith, S. F., Cortellessa, G., Hildum, D. W. and Ohler, C. M. (2004b) Using a scheduling domain ontology to compute user-oriented explanations. In *Planning and Scheduling*, . Castillo, D. Borrajo, M. A. Salido and A. Oddi (Eds.), Frontiers in Artificial Intelligence and Applications Series, IOS Press, Amsterdam, forthcoming.

Smith, S. F., Hildum, D. W. and Crimm, D. (2003) Interactive resource management in the comirem planner. *IJCAI-03 Workshop on Mixed-Initiative Intelligent Systems* (Acapulco, Mexico), pp. 100–106.

Smith, S. F. and Zimmerman, T. L. (2004) Planning tactics within scheduling problems. In *Proceedings ICAPS-04 Workshop on Integrating Planning into Scheduling* (Whistler, BC), pp. 83–90.

Srivastava. B., Kambhampati, S. and Minh, B. D. (2001) Planning the project management way: efficient planning by effective integration of causal and resource reasoning in RealPlan. *Artificial Intelligence*, **131**:73–134.

Sycara, K., Roth, S. F., Sadeh, N. and Fox, M. S. (1991) Resource allocation in distributed factory scheduling. *IEEE Expert*, **6**:29–40.

Syswerda, G. (1991) Schedule optimization using genetic algorithms. In *Handbook of Genetic Algorithms*, L.D. Davis (Ed.), Van Nostrand-Reinhold, New York.

Talukdar, S., Baerentzen, L., Gove, A. and de Souza, P. (1998) Asynchronous teams: cooperation schemes for autonomous agents. *Journal of Heuristics*, **4**:295–321.

T'kindt, V. and Billaut, J. C. (2002) *Multicriteria Scheduling*, Springer, Berlin.

Voss, S. (2001) Meta-heuristics: the state of the art. In *Local Search for Planning and Scheduling*, A. Nareyek (Ed.), Lecture Notes in Artificial Intelligence, Vol. 2148, Springer, Berlin, pp. 1–23.

Wagner, T., Phelps, J., Guralnik, V. and VanRiper, R. (2004) COORDINATORS: coordination managers for first responders. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems* (New York).

Wallace, R. and Freuder, E. (2000) Dispatchable Execution of schedules involving consumable resources. In *Proceedings of the 5th International Conference on AI Planning and Scheduling* (Breckenridge CO), AAAI Press, Menlo Park, CA, pp. 283–290.

Walsh, T. (1997) Depth-bounded discrepancy search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pp. 1388–1395.

Wang, X. and Smith, S. F. (2004) Generating schedules to maximize quality. *Robotics Institute Technical Report CMU-RI-TR-04-25*, Carnegie Mellon University, Pittsburgh, PA.

Watson, J.-P., Barbulescu, L., Howe, A. E., and Whitley, L. D. (1999) Algorithm performance and problem structure for flow-shop scheduling. In *Proceedings of the 16th National Conference on Artificial Intelligence* (Orlando, FL), pp. 688–695.

Watson, J.-P. (2003) Empirical modeling and analysis of local search algorithms for the job-shop scheduling problem. *Ph.D. Thesis*, Department of Computer Science, Colorado State University.

Wellman, M. P., Walsh, W. E., Wurman, P. R. and MacKie-Mason, J. K. (2001) Auction protocols for decentralized scheduling. *Games and Economic Theory*, **35**:271–303.

Zhou, Q. and Smith, S. F. (2002) A priority-based preemption algorithm for incremental scheduling with cumulative resources. *Robotics Institute Technical Report* CMU-RI-TR-02-19, Carnegie Mellon University.

Zweben, M., Daun, B., Davis, E. and Deale, M. (1994) Scheduling and rescheduling with iterative repair. In *Intelligent Scheduling*, M. Zweben and M. Fox (Eds.), Morgan Kaufmann, San Mateo, CA.

FORMULATIONS, RELAXATIONS, APPROXIMATIONS, AND GAPS IN THE WORLD OF SCHEDULING

Gerhard J. Woeginger
Department of Mathematics and Computer Science, TU Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
g.j.woeginger@tue.nl

Abstract We discuss a number of polynomial time approximation results for scheduling problems. All presented results are based on the technique of rounding the optimal solution of an underlying linear programming relaxation. We analyse these relaxations, their integrality gaps, and the resulting approximation algorithms, and we derive matching worst-case instances.

Keywords: approximation algorithm, worst-case analysis, performance guarantee, linear programming relaxation, integrality gap, scheduling.

INTRODUCTION

Most real-world optimization problems are NP-hard, and most NP-hard problems are difficult to solve to optimality. We conclude: most real-world problems are difficult to solve to optimality. A standard way of working around this (rather pessimistic) conclusion is to forget about *exact optimality*, and to satisfy oneself instead with *near-optimal* or *approximate* solutions. This leads us into the area of approximation algorithms for combinatorial optimization problems.

A combinatorial optimization problem consists of a set $\mathcal{I}$ of instances, and a family $\mathcal{F}(I)$ of feasible solutions for every instance $I \in \mathcal{I}$. Every feasible solution $F \in \mathcal{F}(I)$ comes with a non-negative cost $c(F)$. In this paper, we will only consider minimisation problems, where the objective is to determine a feasible solution of minimum possible cost. An *approximation algorithm* is an algorithm that for every instance $I \in \mathcal{I}$ returns a near-optimal solution. If it manages to do this in polynomial time, then it is called a *polynomial time* approximation algorithm. An approximation algorithm for a minimisation problem is called a *ρ-approximation algorithm*, if it always returns a near-optimal

solution with cost at most a factor ρ above the optimal cost. Such a value $\rho \geq 1$ is called a *worst-case performance guarantee* of the algorithm.

Approximations through relaxations. One standard approach for designing polynomial time approximation algorithms for a (difficult, NP-hard) optimisation problem $\mathcal{P}$ is the following:

(S1) Relax some of the constraints of the hard problem $\mathcal{P}$ to get an easier problem $\mathcal{P}'$ (the so-called *relaxation*).

(S2) Compute (in polynomial time) an optimal solution S' for this easier relaxed problem $\mathcal{P}'$.

(S3) Translate (in polynomial time) the solution S' into an approximate solution S for the original problem $\mathcal{P}$.

(S4) Analyse the quality of solution S for $\mathcal{P}$ by comparing its cost to the cost of solution S' for $\mathcal{P}'$.

Let C^{Opt} denote the optimal cost of the original problem instance, let C^{Rlx} denote the optimal cost of the relaxed instance, and let C^{App} denote the cost of the translated approximate solution. To show that the sketched approach has a performance guarantee of ρ, one usually establishes the following chain of inequalities:

$$C^{Rlx} \leq C^{Opt} \leq C^{App} \leq \rho \cdot C^{Rlx} \leq \rho \cdot C^{Opt} \tag{1}$$

The first and the last inequality in this chain are trivial, since problem $\mathcal{P}'$ results from problem $\mathcal{P}$ by relaxing constraints. The second inequality is also trivial, since the optimal solution is at least as good as some approximate solution. The third inequality contains the crucial step in the chain, and all the analysis work goes into proving that step. This third inequality relates the relaxed solution to the approximate solution; both solutions are polynomial time computable, and hence their combinatorics will be nice and well-behaved. Thus, the chain yields the desired relation $C^{App} \leq \rho \cdot C^{Opt}$ by analysing nice, polynomial time computable objects. The analysis avoids touching the original NP-hard problem whose combinatorics is messy and complicated and hard to grasp.

Worst-case gaps and integrality gaps. Of course, we would like to make the value of the parameter ρ as small as possible: The closer ρ is to 1, the better is the performance of the approximation algorithm. How can we argue that our worst-case analysis is complete? How can we argue that we have reached the smallest possible value for ρ? That is usually done by exhibiting a so-called *worst-case instance*, that is, an instance I that demonstrates a *worst-case gap* of ρ for the approximation algorithm:

$$C_I^{App} = \rho \cdot C_I^{Opt} \qquad \text{and} \qquad C_I^{Opt} = C_I^{Rlx} \tag{2}$$

Here the left-hand equation establishes the gap, and together with the chain (1) it yields the right-hand equation. The worst-case instance (2) illustrates that our analysis of the *combined* approach (S1)–(S3) is tight. Is this the end of the story? Not necessarily. We could possibly start with the same relaxation step (S1), then solve the relaxation with the same step (S2), and then come up with a completely new (and better) translation step. How can we argue that this is not possible? How can we argue that the value ρ is already the best performance guarantee that we possibly can get out of the considered relaxation? That is usually done by exhibiting an instance J that demonstrates an *integrality gap* of ρ between the original problem and the relaxation:

$$C_J^{Opt} = \rho \cdot C_J^{Rlx} \qquad \text{and} \qquad C_J^{Opt} = C_J^{App} \tag{3}$$

The equation on the left-hand side establishes the gap, and with (1) it yields the equation on the right-hand side. In particular, we have $C_J^{App} = \rho \cdot C_J^{Rlx}$. For instance J the third inequality in the chain (1) is tight, and there is no way of proving a better performance guarantee for an approximation algorithm built around the considered relaxation. We stress that such a better approximation algorithm around the relaxation might well exist, but we will never be able to *prove* that its performance guarantee is better than ρ within the framework described above.

For $\rho > 1$, the conditions in (2) and in (3) cannot be satisfied by the same instance, as they would be contradictory. Hence, a complete analysis of an approximation algorithm within our framework always must provide *two* separate bad instances, one for the worst-case gap and one for the integrality gap.

Overview of this paper. We will illustrate the approach (S1)–(S4) with three examples from scheduling theory presented in the following three sections. For each example we provide an integer programming formulation, a relaxation, an approximation algorithm, a worst-case analysis, and two gap instances. In the conclusions section we give some pointers to the literature, and we pose one open problem.

Throughout the paper, we use the standard three-field scheduling notation (see e.g. Graham *et al.*, 1979; Lawler *et al.*, 1993).

1. MAKESPAN ON PARALLEL MACHINES

As our first example we will study $P\,||\,C_{\max}$, the problem of minimising makespan on parallel identical machines. The input consists of m identical machines $M_1, \ldots, M_m$ together with n jobs $J_1, \ldots, J_n$ with processing times $p_1, \ldots, p_n$. Every job is available for processing at time 0. The goal is to schedule the jobs such that the maximum job completion time (the so-called *makespan* $C_{\max}$) is minimised. Problem $P\,||\,C_{\max}$ is known to be NP-hard in

the strong sense (Garey and Johnson, 1979). The goal of this section is to give a first, simple illustration for the topics discussed in this paper.

Exact formulation and relaxation. Consider the following integer programming formulation (4) of problem $P\,||\,C_{\max}$. For machine M_i and job J_j, the binary variable x_{ij} decides whether J_j is assigned to M_i (in which case $x_{ij} = 1$) or whether J_j is assigned to some other machine (in which case $x_{ij} = 0$). The continuous variables L_i describe the total load (i.e. the total job processing time) assigned to machine M_i. Finally, the continuous variable C denotes the makespan:

$$\begin{array}{lll} \min & C & \\ \text{s.t.} & \sum_{i=1}^{m} x_{ij} = 1 & \text{for } j = 1,\ldots,n \\ & \sum_{j=1}^{n} x_{ij} p_j = L_i & \text{for } i = 1,\ldots,m \\ & L_i \leq C & \text{for } i = 1,\ldots,m \\ & p_j \leq C & \text{for } j = 1,\ldots,n \\ & x_{ij} \in \{0,1\} & \text{for } i = 1,\ldots,m \text{ and } j = 1,\ldots,n \end{array} \tag{4}$$

The first family of constraints states that every job has to be assigned to precisely one machine. The second family of constraints connects the assigned jobs to the machine loads, and the third family of constraints connects the machines loads to the makespan. The fourth family of constraints requires that the makespan is at least as large as any job processing time.

Since $P\,||\,C_{\max}$ is an NP-hard problem, the equivalent integer programming formulation (4) will also be NP-hard to solve. Therefore, we relax this formulation to get something simpler: We replace the integrality constraints "$x_{ij} \in \{0,1\}$" by continuous constraints "$0 \leq x_{ij} \leq 1$". Now all the variables are continuous variables, and so this relaxation is a linear program that can be solved to optimality in polynomial time. From now on we will use $P = \sum_{j=1}^{n} p_j$ to denote the overall job processing time. Furthermore, we denote the optimal objective value of (4) by C^{Opt}, and the optimal objective value of the LP-relaxation by C^{LP}. Setting $x_{ij}^{LP} \equiv 1/m$ and $L_i^{LP} \equiv P/m$ gives us a feasible solution for the LP-relaxation. In fact, it can be verified that this feasible solution is an optimal LP-solution with objective value

$$C^{LP} = \max\left\{\frac{1}{m}P,\ \max_{j=1}^{m} p_j\right\} \tag{5}$$

The approximation algorithm. Now let us design a polynomial time approximation algorithm for $P\,||\,C_{\max}$ that is based on the above ideas. Since

C^{LP} is an under-estimation of the true optimal objective value C^{Opt}, we will multiply C^{LP} by an appropriate stretching factor $\beta := 2m/(m+1)$ and reserve a time interval of length $\beta \cdot C^{LP}$ on each machine:

- Turn every machine into a corresponding bin of capacity $\beta \cdot C^{LP}$.
- Pack the processing times $p_1, \ldots, p_n$ one by one into these m bins. Every processing time is packed into the first bin (the bin with smallest index) into which it will fit.

That is a very simple and fairly natural algorithm. In the case all n processing times can be fit into the bins, the makespan of the corresponding schedule is at most

$$C^{App} \leq \beta \cdot C^{LP} \leq \beta \cdot C^{Opt} = \frac{2m}{m+1} \cdot C^{Opt}$$

This would yield a worst-case performance guarantee of $\beta = 2m/(m+1)$. Hence, it remains to be shown that we indeed can pack all the jobs into the bins. Suppose for the sake of contradiction that this is not possible. Consider the first moment in time where some job, say job J_y with processing time p_y, does not fit into any bin. Let $B_1, \ldots, B_m$ be the contents of the bins at that moment, and let $B_x = \min_i B_i$ denote the smallest contents. Since p_y does not fit into any bin,

$$p_y + B_i > \beta \cdot C^{LP} \qquad \text{for } i = 1, \ldots, m \tag{6}$$

In particular, this implies that all bins are non-empty. Next, we claim that

$$B_i + B_x > \beta \cdot C^{LP} \qquad \text{for } i = 1, \ldots, m \text{ with } i \neq x \tag{7}$$

Indeed, if $i < x$ then every job in B_x was too big to fit into B_i, and if $i > x$ then every job in B_i was too big to fit into B_x. This proves (7). Adding up the m inequalities in (6) yields

$$m \cdot p_y + \sum_{i=1}^{m} B_i > m \cdot \beta \cdot C^{LP} \tag{8}$$

Since $p_y + \sum_{i=1}^{m} B_i \leq P$, the left hand side in (8) is bounded from above by $P + (m-1)p_y$, which by (5) in turn is bounded from above by $m \cdot C^{LP} + (m-1)p_y$. Plugging this upper bound into (8) and simplifying yields that

$$p_y > \frac{\beta}{2} \cdot C^{LP} \tag{9}$$

By adding the $m-1$ inequalities in (7) to the inequality (9), we get that

$$p_y + (m-2)B_x + \sum_{i=1}^{m} B_i > \left(m - \frac{1}{2}\right) \cdot \beta \cdot C^{LP} \tag{10}$$

The left-hand side in (10) is bounded from above by $P + (m-2)B_x$, which by (5) in turn is bounded from above by $m\,C^{LP} + (m-2)B_x$. Plugging this upper bound into (8) and simplifying yields that

$$B_x \;>\; \frac{\beta}{2} \cdot C^{LP} \tag{11}$$

Combining (9) with (11) leads to

$$P \;\geq\; p_y + \sum_{i=1}^{m} B_i \;>\; (m+1) \cdot \frac{\beta}{2} \cdot C^{LP} \;=\; m \cdot C^{LP} \tag{12}$$

Since this blatantly contradicts (5), we have arrived at the desired contradiction. To summarise, the described bin packing algorithm indeed has a worst-case performance guarantee of at most $\beta = 2m/(m+1)$.

Analysis of the two gaps. Is our worst-case bound of β for the bin packing algorithm best possible? Yes, it is; consider the instance that consists of m jobs of length $1/(m+1)$ followed by m jobs of length $m/(m+1)$. Then $C^{Opt} = 1$, whereas $C^{App} = \beta$. (This instance also illustrates that for stretching factors strictly less than β, the bin packing algorithm does not necessarily end up with a feasible solution.)

Gap 1.1 *There exist instances for $P\,||\,C_{\max}$ for which the gap between the optimal makespan C^{Opt} and the makespan produced by the bin packing algorithm equals $\beta = 2m/(m+1)$.*

What about the integrality gap of the LP-relaxation? Consider the instance with $n = m+1$ jobs and processing times $p_j \equiv m/(m+1)$. Since the optimal solution must put some two of these $m+1$ jobs on the same machine, we have $C^{Opt} \geq 2m/(m+1)$. For the LP-relaxation, the formula in (5) yields $C^{LP} = 1$.

Gap 1.2 *The integrality gap of the linear programming relaxation for problem $P\,||\,C_{\max}$ is $\beta = 2m/(m+1)$. For large m, the gap goes to 2.*

The results derived in this section for $P\,||\,C_{\max}$ are fairly weak. The main motivation for stating them was to illustrate the basic relaxational approach. The literature contains much stronger approximation results for $P\,||\,C_{\max}$. Already back in the 1960s, Graham (1966, 1969) investigated simple greedy algorithms with worst-case performance guarantees $4/3 - 1/(3m)$. In the

1980s, Hochbaum and Shmoys (1987) designed a *polynomial time approximation scheme* for $P\,||\,C_{\max}$: for every $\varepsilon > 0$, they provide a polynomial time approximation algorithm with worst-case performance guarantee at most $1+\varepsilon$.

The LP-relaxation of (4) essentially describes the *preemptive* version $P\,|\,pmtn\,|\,C_{\max}$ of makespan minimisation, and the formula in (5) states the optimal preemptive makespan. Woeginger (2000) discusses preemptive versus non-preemptive makespan for *uniform* machines (that is, machines that run at different speeds). Woeginger (2000) shows that the integrality gap between $Q\,||\,C_{\max}$ and $Q\,|\,pmtn\,|\,C_{\max}$ is at most $2 - 1/m$, and that this bound is the best possible. Lenstra *et al.* (1990) discuss similar relaxations for the problem $R\,||\,C_{\max}$ with unrelated machines. They establish an upper bound of 2 for the integrality gap of the preemptive relaxation.

2. MAKESPAN UNDER COMMUNICATION DELAYS

Communication delays (see Papadimitriou and Yannakakis, 1990; Veltman *et al.*, 1990) take the data transmission times between machines into account. We will look at one of the simplest problems in this area, where all jobs have unit processing times and where all the communication delays are one time-unit long.

There are n unit-length jobs $J_1, \ldots, J_n$ that are precedence constrained in the following way: if there is a precedence constraint $J_i \rightarrow J_j$ between jobs J_i and J_j, then job J_j cannot be started before job J_i has been completed. Furthermore, if jobs J_i and J_j are processed on *different* machines, then it takes one time-unit to transmit the data generated by job J_i over to job J_j; in this case, J_j cannot start earlier than one time-unit after the completion time of J_i. The number of machines is not a bottleneck, and may be chosen by the scheduler. All jobs are available at time 0, and the objective is to minimise the makespan.

This problem is denoted by $P\infty\,|\,prec, p_j{=}1, c{=}1\,|\,C_{\max}$. Picouleau (1992) has proved its NP-hardness.

Exact formulation and relaxation. We introduce the notation $\textsc{Succ}(j) = \{i : J_j \rightarrow J_i\}$ and $\textsc{Pred}(j) = \{i : J_i \rightarrow J_j\}$ to encode the successor and predecessor sets of job J_j. Consider the following integer programming formulation (13) of problem $P\infty\,|\,prec, p_j{=}1, c{=}1\,|\,C_{\max}$. The continuous variable C_j denotes the completion time of job J_j. For every pair of jobs $J_i \rightarrow J_j$, the binary variable x_{ij} decides whether there is a unit-time communication delay between J_i and J_j (in which case $x_{ij} = 1$), or whether there is no delay and J_j is run immediately after J_i on the same machine (in which case $x_{ij} = 0$). The

continuous variable C denotes the makespan:

$$\begin{array}{lll} \min & C & \\ \text{s.t.} & \sum_{i \in Pred(j)} x_{ij} \geq |\text{PRED}(j)| - 1 & \text{for } j = 1, \ldots, n \\ & \sum_{i \in Succ(j)} x_{ji} \geq |\text{SUCC}(j)| - 1 & \text{for } j = 1, \ldots, n \\ & C_i + 1 + x_{ij} \leq C_j & \text{for } J_i \rightarrow J_j \\ & 1 \leq C_j \leq C & \text{for } j = 1, \ldots, n \\ & x_{ij} \in \{0, 1\} & \text{for } J_i \rightarrow J_j \end{array} \tag{13}$$

Consider some fixed job J_j that completes at time C_j on machine M_z in some fixed feasible schedule. All predecessors of J_j that are processed on machines $\neq M_z$ must complete at time $C_j - 2$ or earlier. And all predecessors of J_j that are processed on machine M_z (with the exception of the last predecessor) must also complete at time $C_j - 2$ or earlier. Hence, at most one of the communication delays x_{ij} with $i \in \text{PRED}(j)$ can be 0, and all the others can be set to 1. All this is expressed by the first family of constraints. The second family of constraints states a symmetric condition for the successor sets. The third family of constraints connects the job completion times to the communication delays, and the fourth family of constraints connects the job completion times to the makespan.

Next, we will relax the integer programming formulation by replacing the integrality constraints "$x_{ij} \in \{0, 1\}$" by continuous constraints "$0 \leq x_{ij} \leq 1$". We get the corresponding LP-relaxation that can be solved in polynomial time. We denote an optimal solution of this LP by x_{ij}^{LP}, C_j^{LP}, and C^{LP}.

The approximation algorithm. Now let us translate the LP-solution into a "nearly" feasible IP-solution. The trouble-makers in the LP-solution are the delay variables x_{ij}^{LP} that need not be integral, whereas we would like them to be binary. A simple way of resolving this problem is threshold rounding: if $x_{ij}^{LP} < 1/2$, then we define a *rounded* variable $\tilde{x}_{ij} = 0$, and if $x_{ij}^{LP} \geq 1/2$, then we define a rounded variable $\tilde{x}_{ij} = 1$. Then we run through the jobs and fix the job completion times; the jobs are handled in topological order (that is, we handle a job only *after* all of its predecessors already have been handled). For a job J_j without predecessors, we define a *rounded* job completion time $\tilde{C}_j = 1$. For a job with predecessors, we define a rounded completion time

$$\tilde{C}_j \;=\; \max\{\tilde{C}_i + 1 + \tilde{x}_{ij} : \; i \in \text{PRED}(j)\} \tag{14}$$

That is, we place every job at the earliest possible moment in time without violating the precedence constraints and communication delays. Finally, we compute the *rounded* makespan $\tilde{C} = \max_j\{\tilde{C}_j\}$. Altogether, this gives us a *rounded* solution for (13).

Let us verify that the rounded values $\tilde{x}_{ij}$, $\tilde{C}_j$, and $\tilde{C}$ constitute a feasible solution of (13): if they violate a constraint from the first family of constraints, then $\tilde{x}_{ij} = \tilde{x}_{kj} = 0$ must hold for some $i, k \in \text{PRED}(j)$. But this would mean $x_{ij}^{LP} < 1/2$ and $x_{kj}^{LP} < 1/2$, whereas for all other $\ell \in \text{PRED}(j)$ we have $x_{\ell j}^{LP} \leq 1$. Consequently,

$$\sum_{i \in Pred(j)} x_{ij}^{LP} \;<\; \frac{1}{2} + \frac{1}{2} + (|\text{PRED}(j)| - 2) \cdot 1 \;=\; |\text{PRED}(j)| - 1$$

and $x_{ij}^{LP}, C_j^{LP}, C^{LP}$ would not be feasible for the LP-relaxation; a contradiction. A symmetric argument shows that the rounded solution also satisfies the second family of constraints. The third and fourth family are satisfied by the way we have computed the values $\tilde{C}_j$ and $\tilde{C}$. Hence, the rounded solution is indeed feasible for (13). What about its quality? Let us first observe that

$$1 + \tilde{x}_{ij} \;\leq\; \frac{4}{3}\,(1 + x_{ij}^{LP}) \qquad \text{for all } J_i \rightarrow J_j \tag{15}$$

First, consider the case $\tilde{x}_{ij} = 0$. Then in the inequality (15) the left-hand side equals 1, the right-hand side is at least $4/3$, and the statement clearly is true. Secondly, consider the case $\tilde{x}_{ij} = 1$. Then the inequality (15) is equivalent to $x_{ij}^{LP} \geq 1/2$, and that was precisely the condition for setting $\tilde{x}_{ij} = 1$. These two cases establish the correctness of (15). Next, let us analyse the rounded job completion times. We claim that

$$\tilde{C}_j \;\leq\; \frac{4}{3}\,C_j^{LP} \qquad \text{for all } j = 1, \ldots, n \tag{16}$$

This statement clearly holds true for all jobs without predecessors, since they satisfy $\tilde{C}_j = C_j^{LP} = 1$. For the remaining jobs, we use an inductive argument along the topological ordering of the jobs:

$$\begin{aligned}
\tilde{C}_j \;&=\; \max\left\{\tilde{C}_i + 1 + \tilde{x}_{ij} :\; i \in \text{PRED}(j)\right\} \\
&\leq\; \max\left\{\frac{4}{3}\,C_i^{LP} + \frac{4}{3}\,(1 + x_{ij}^{LP}) :\; i \in \text{PRED}(j)\right\} \\
&=\; \frac{4}{3}\,\max\left\{C_i^{LP} + (1 + x_{ij}^{LP}) :\; i \in \text{PRED}(j)\right\} \\
&\leq\; \frac{4}{3}\,C_j^{LP}
\end{aligned}$$

Here the first equation comes from (14); the first inequality follows from (15) and (16); the second inequality follows from the third family of constraints in (13). Finally, since all rounded job completion times are at most a factor of

4/3 above the job completion times in the LP-solution, the rounded makespan satisfies

$$\tilde{C} \leq \frac{4}{3} C^{LP} \leq \frac{4}{3} C^{Opt}$$

Hence, the described approximation algorithm has a worst-case performance guarantee of at most 4/3. This result is due to Munier & König (1997).

Analysis of the two gaps. Now let us find a worst-case instance for the approximation algorithm: we use $n = 3k + 1$ jobs that we call $J_1, \ldots, J_{k+1}$, and $J'_1, \ldots, J'_k$, and $J''_1, \ldots, J''_k$. For $j = 1, \ldots, k$ we introduce the precedence constraints $J_j \rightarrow J'_j \rightarrow J_{j+1}$ and $J_j \rightarrow J''_j \rightarrow J_{j+1}$.

- An optimal solution sequences all jobs on a single machine such that job J_j completes at time $3j - 2$, job J'_j completes at time $3j - 1$, and job J''_j completes at time $3j$. This yields a makespan of $3k + 1$.
- An optimal LP-solution may set all delay variables $x_{ij}^{LP} \equiv 1/2$. Then the completion time of J_j becomes $3j-2$, the completion times of J'_j and J''_j become $3j - \frac{1}{2}$, and the optimal objective value becomes $C^{LP} = 3k + 1$.
- In the rounded solution, all delay variables are $\tilde{x}_{ij} \equiv 1$. Job J_j completes at time $4j - 3$, and jobs J'_j and J''_j complete at time $4j - 2$. Hence, the rounded makespan is $\tilde{C} = 4k + 1$.

Gap 2.1 *There exist instances for* $P\infty \mid prec, p_j{=}1, c{=}1 \mid C_{\max}$ *for which the gap between the optimal makespan and the makespan produced by the rounding algorithm comes arbitrarily close to* 4/3.

And what about the integrality gap of the LP-relaxation for problem $P\infty \mid prec,\ p_j{=}1, c{=}1 \mid C_{\max}$? Consider an instance with $2^k - 1$ jobs, where the precedence constraints form a complete binary out-tree of height $k - 1$. That is, every job (except the leaves) has exactly two immediate successors, and every job (except the root) has exactly one immediate predecessor.

- Consider a non-leaf job J_j that completes at time C_j in the optimal solution. Only one of its two immediate successors can be processed on the same machine during the time slot $[C_j; C_j + 1]$, whereas the other immediate successor must complete at time $C_j + 2$. This yields that the optimal makespan equals $2k - 1$.
- Now consider an LP-solution in which all delay variables are $x_{ij}^{LP} \equiv 1/2$. Then jobs at distance d from the root complete at time $1 + \frac{3}{2}d$. Therefore, $C^{LP} = \frac{1}{2}(3k - 1)$.

Gap 2.2 *For problem* $P\infty \mid prec, p_j{=}1, c{=}1 \mid C_{\max}$, *the integrality gap of the LP-relaxation is* $4/3$.

The approximation result described in this section is the strongest known result for $P\infty \mid prec, p_j{=}1, c{=}1 \mid C_{\max}$ (Munier and König, 1997). It is an outstanding open problem to decide whether there exists a better polynomial time approximation algorithm, with worst-case performance guarantee strictly less than $4/3$.

The literature contains a remarkable *negative* result in this direction: Hoogeveen *et al.* (1994) have shown that the existence of a polynomial time approximation algorithm with worst-case performance guarantee strictly less than $7/6$ would imply $P = NP$.

3. PREEMPTIVE MAKESPAN UNDER JOB REJECTIONS

In this section, we consider an environment with n jobs $J_1, \ldots, J_n$ and with m *unrelated* parallel machines $M_1, \ldots, M_m$.

Job J_j has a processing time p_{ij} on machine M_i, and moreover job J_j has a positive *rejection penalty* f_j. All jobs are available at time 0. Preemption of the jobs is allowed (that is, a job may be arbitrarily interrupted and resumed later on an arbitrary machine). A job may be processed on at most one machine at a time. For each job J_j, it must be decided whether to accept or to reject it. The accepted jobs are to be scheduled on the m machines. For the accepted jobs, we pay the makespan of this schedule. For the rejected jobs, we pay their rejection penalties. In other words, the objective value is the preemptive makespan of the accepted jobs plus the total penalty of the rejected jobs.

This scheduling problem is denoted by $R \mid pmtn \mid \text{Rej} + C_{\max}$; it is NP-hard in the strong sense (Hoogeveen *et al.*, 2003).

Exact formulation and relaxation. Once again, we will start by stating an integer programming formulation. For job J_j, the binary variable y_j decides whether J_j is rejected ($y_j = 0$) or accepted ($y_j = 1$). The continuous variables x_{ij} describe which percentage of job J_j should be processed on machine M_i. The continuous variable C denotes the optimal preemptive makespan for the

accepted jobs:

$$\begin{array}{lll} \min & C + \sum_{j=1}^{n}(1-y_j)f_j & \\ \text{s.t.} & \sum_{j=1}^{n} x_{ij}p_{ij} \leq C & \text{for } i = 1,\ldots,m \\ & \sum_{i=1}^{m} x_{ij}p_{ij} \leq C & \text{for } j = 1,\ldots,n \\ & \sum_{i=1}^{m} x_{ij} = y_j & \text{for } j = 1,\ldots,n \\ & x_{ij} \geq 0 & \text{for } i = 1,\ldots,m \text{ and } j = 1,\ldots,n \\ & y_j \in \{0,1\} & \text{for } j = 1,\ldots,n \end{array} \tag{17}$$

The first family of constraints states that for every machine the total assigned processing time is at most C. The second family of constraints states that the total processing time of any accepted job cannot exceed C. The third family of constraints connects the binary decision variables y_j to the continuous variables x_{ij}: if a job is accepted ($y_j = 1$), then the percentages x_{ij} should add up to $1 = y_j$; if a job is rejected ($y_j = 0$), then the percentages x_{ij} should add up to $0 = y_j$.

As soon as we have fixed all the values x_{ij}, the remaining makespan minimisation problem is essentially makespan minimisation in a preemptive open shop. It is well known (Gonzalez and Sahni, 1976; Lawler *et al.*, 1993) that for a preemptive open shop, the smallest value C fulfilling the first and second family of constraints in (17) yields the optimal preemptive makespan. To summarise, the integer program (17) is a complete and correct description of the problem $R \,|\, pmtn \,|\, \text{Rej} + C_{\max}$.

We define the LP-relaxation of the integer programming formulation (17) by replacing the integrality constraints "$y_j \in \{0,1\}$" by continuous constraints "$0 \leq y_j \leq 1$". This LP-relaxation can be solved in polynomial time, and we denote an optimal solution by x_{ij}^{LP}, y_j^{LP}, and C^{LP}.

The approximation algorithm. Now we want to translate the LP-solution into a reasonable feasible solution for (17). We mainly have to take care of the decision variables y_j; however, this time we also must pay attention to the continuous variables x_{ij}, since their values depend on the values y_j via the third family of constraints in (17).

We *randomly* choose a threshold α from the uniform distribution over the interval $[1/\mathrm{e}, 1]$; here as usual $\mathrm{e} \approx 2.71828$ denotes the base of the natural logarithm. If $y_j^{LP} \leq \alpha$, then we define a rounded decision variable $\tilde{y}_j := 0$, and otherwise we define $\tilde{y}_j := 1$. Jobs J_j with $\tilde{y}_j = 0$ are rejected in the rounded solution, and we set all their variables $\tilde{x}_{ij} = 0$. Jobs J_j with $\tilde{y}_j = 1$ are accepted in the rounded solution; we set all their variables $\tilde{x}_{ij} := x_{ij}^{LP}/y_j^{LP}$.

Finally, we define the rounded makespan by

$$\tilde{C} := \max\left\{ \max_{1\le i\le m} \sum_{j=1}^{n} \tilde{x}_{ij} p_{ij},\ \max_{1\le j\le n} \sum_{i=1}^{m} \tilde{x}_{ij} p_{ij} \right\} \tag{18}$$

It can be verified that the rounded solution $\tilde{x}_{ij}$, $\tilde{y}_j$, and $\tilde{C}$ constitutes a feasible solution of (17): all variables $\tilde{y}_j$ are binary. For j with $\tilde{y}_j = 0$, the variables $\tilde{x}_{ij}$ add up to 0. For j with $\tilde{y}_j = 1$, the variables $\tilde{x}_{ij}$ add up to $\sum_i x_{ij}^{LP}/y_j^{LP} = 1$. Finally, in (18) the value of $\tilde{C}$ is fixed in order to fulfil the first and the second family of constraints.

Now let us analyse the quality of this rounded solution. For any fixed value of α, the rounded variable $\tilde{x}_{ij}$ is at most a factor of $1/\alpha$ above x_{ij}^{LP}. Hence, by linearity also $\tilde{C}$ is at most a factor of $1/\alpha$ above C^{LP}. Then the expected multiplicative increase in the makespan is at most a factor of

$$\frac{\mathrm{e}}{\mathrm{e}-1} \int_{1/\mathrm{e}}^{1} 1/\alpha \, \mathrm{d}\alpha = \frac{\mathrm{e}}{\mathrm{e}-1}$$

In the LP-solution, the contribution of job J_j to the total rejection penalty is $(1 - y_j^{LP}) f_j$. The expected contribution of J_j to the rejection penalty in the rounded solution is

$$\begin{aligned} f_j \cdot \mathrm{Prob}[y_j^{LP} \le \alpha] &= f_j \int_{\max\{1/\mathrm{e}, y_j^{LP}\}}^{1} \frac{\mathrm{e}}{\mathrm{e}-1} \mathrm{d}\alpha \\ &\le f_j \int_{y_j^{LP}}^{1} \frac{\mathrm{e}}{\mathrm{e}-1} \mathrm{d}\alpha \\ &= \frac{\mathrm{e}}{\mathrm{e}-1} \cdot (1 - y_j^{LP}) f_j \end{aligned}$$

All in all, the expected objective value for the rounded solution is at most a factor of $\mathrm{e}/(\mathrm{e}-1) \approx 1.58$ above the optimal objective value of the LP-relaxation. Hence, our procedure yields a *randomised* polynomial time approximation algorithm with a worst-case performance guarantee of $\mathrm{e}/(\mathrm{e}-1)$.

How can we turn this randomised algorithm into a deterministic algorithm? Well, the only critical values for the threshold parameter α are the values y_j^{LP} with $j = 1, \ldots, n$. All other values of α will yield the same solution as for one of these critical values. Hence, it is straightforward to derandomise the algorithm in polynomial time: we compute the n rounded solutions that correspond to these n critical values, and we select the solution with smallest objective value.

Analysis of the two gaps. Our next goal is to give a worst-case instance for the above approximation algorithm for $R \,|\, pmtn \,|\, \mathrm{Rej} + C_{\max}$. The instance

is based on an integer parameter q. There are $m = (q+1)^q - q^q$ machines M_j that are indexed by $j = q^q + 1, \ldots, (q+1)^q$. For every machine M_j, there are two corresponding jobs J_j and J'_j that have infinite processing requirements on all other machines M_i with $i \neq j$; this implies that these two jobs either have to be rejected or have to be processed on M_j. The processing time of job J_j on M_j is $j - q^q$, and its rejection penalty is $(j - q^q)/j$. The processing time of job J'_j on M_j is q^q, and its rejection penalty is q^q/j. Note that the overall processing time of J_j and J'_j is j, and that their overall penalty is 1.

One possible feasible solution accepts all the jobs J'_j and rejects all the jobs J_j. The resulting makespan is $C = q^q$, and the resulting objective value equals

$$q^q + \sum_{j=q^q+1}^{(q+1)^q} (j - q^q) \frac{1}{j} \;=\; (q+1)^q - q^q \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \tag{19}$$

It can be verified that this in fact is the optimal objective value. Next, assume that the approximation algorithm starts from the following feasible solution for the LP-relaxation: for $j = q^q + 1, \ldots, (q+1)^q$ the two jobs J_j and J'_j both get an acceptance value $y_j^{LP} = q^q/j$. Then on every machine M_j, the overall accepted processing time is $(j - q^q)y_j^{LP} + q^q\, y_j^{LP} = q^q$. The penalty of job J_j plus the penalty of job J'_j equals $1 - y_j^{LP} = (j - q^q)/j$. Hence, the objective value of this LP-solution equals the value in (19).

Consider the rounding step for some fixed threshold α. Since the values $y_j^{LP} = q^q/j$ are decreasing in j, there exists some index k such that for $j \leq k$ the values $y_j^{LP} = q^q/j$ all are rounded up to 1 (and the corresponding jobs are accepted), whereas for $j \geq k+1$ the values $y_j^{LP} = q^q/j$ all are rounded down to 0 (and the corresponding jobs are rejected). Then the makespan becomes k (the load on machine M_k), the total rejection penalty is $(q+1)^q - k$, and the objective value is $(q+1)^q$. Thus, the objective value in the rounded solution always equals $(q+1)^q$, and does not depend on α or k.

The ratio between the optimal objective value in (19) and the approximate objective value of $(q+1)^q$ equals

$$1 - \left(\frac{q}{q+1}\right)^q \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \tag{20}$$

We estimate the sum in (20) by

$$\int_{q^q+1}^{(q+1)^q+1} \frac{1}{z}\,\mathrm{d}z \;\leq\; \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \;\leq\; \int_{q^q}^{(q+1)^q} \frac{1}{z}\,\mathrm{d}z$$

Since the integrals on the left- and on the right-hand side both converge to 1 when q goes to infinity, the same holds true for the sum in between. Hence, the

ratio in (20) behaves roughly like $((q+1)^q - q^q)/(q+1)^q$. For large q, this ratio tends to $(\mathrm{e}-1)/\mathrm{e}$.

Gap 3.1 *There exist instances for* $R \,|\, pmtn \,|\, \mathrm{Rej} + C_{\max}$ *for which the gap between the optimal makespan and the makespan produced by the rounding algorithm comes arbitrarily close to* $\mathrm{e}/(\mathrm{e}-1)$.

Our final goal in this section is to get the matching lower bound of $\mathrm{e}/(\mathrm{e}-1)$ for the integrality gap of the LP-relaxation for $R \,|\, pmtn \,|\, \mathrm{Rej} + C_{\max}$. We use a slight modification of the instance constructed above. Again, we use an integer parameter q, and again there are $m = (q+1)^q - q^q$ machines M_j that are indexed by $j = q^q + 1, \ldots, (q+1)^q$. For every machine M_j, there is one corresponding job J_j. The processing requirements of J_j are $p_{jj} = j$, and $p_{ij} = \infty$ for $i \neq j$. The rejection penalty is uniformly $f_j \equiv 1$.

Consider a "reasonable" feasible schedule with makespan T: then the jobs on machines M_j with $j \leq T$ will be accepted, and the jobs on machines M_j with $j > T$ will be rejected. This yields a total rejection penalty of $(q+1)^q - T$, and an optimal objective value of $(q+1)^q$. Next, consider the following feasible solution for the LP-relaxation: we set $y_j^{LP} = q^q/j$ for $j = q^q+1, \ldots, (q+1)^q$, and we set $C^{LP} = q^q$. The objective value of this solution is equal to

$$q^q + \sum_{j=q^q+1}^{(q+1)^q} (1 - y_j^{LP}) \quad = \quad (q+1)^q - q^q \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j}$$

Since this LP-value is equal to the value in (19), and since the optimal value is equal to $(q+1)^q$, the ratio of these two values equals the ratio in (20). The arguments around (20) show that as q becomes large, this ratio tends to $(\mathrm{e}-1)/\mathrm{e}$.

Gap 3.2 *For problem* $R \,|\, pmtn \,|\, \mathrm{Rej} + C_{\max}$, *the integrality gap of the LP-relaxation is* $\mathrm{e}/(\mathrm{e}-1)$.

The results presented in this section are essentially due to Hoogeveen *et al.* (2003). It would be nice to get a polynomial time approximation algorithm with a worst-case performance guarantee better than $\mathrm{e}/(\mathrm{e}-1)$. Hoogeveen *et al.* (2003) also show that problem $R \,|\, pmtn \,|\, \mathrm{Rej} + C_{\max}$ is APX-hard; this implies that unless $P = NP$, this problem cannot possess a polynomial time approximation scheme.

4. CONCLUDING REMARKS

We have discussed approximations through relaxations for three scheduling problems. For all three problems, we have presented a complete worst-case

analysis of a polynomial time approximation algorithm. For all three problems, we gave instances that illustrate the worst-case behaviour of the approximation algorithm (the worst-case gap), and we gave instances that illustrate the integrality gap of the linear programming relaxation. For all three problems, the worst-case gap and the integrality gap coincided. The scheduling literature contains many other results that fit into this framework.

- For $1 \,|\, prec \,|\, \sum w_j C_j$, the problem of minimising the total weighted job completion time on a single machine under precedence constraints, the literature contains three different LP-relaxations: one by Potts (1980), one by Dyer and Wolsey (1990), and one by Chudak and Hochbaum (1999). Hall *et al.* (1997) and Chudak and Hochbaum (1999) show that these LP-relaxations have integrality gaps of at most 2. Chekuri and Motwani (1999) design instances that yield matching lower bounds of 2.

- Chekuri *et al.* (2001) investigate the preemptive relaxation for $1 \,|\, r_j \,|\, \sum w_j C_j$, the problem of minimising the total weighted job completion time on a single machine under job release dates. They show that the integrality gap of the preemptive relaxation is at most $\mathrm{e}/(\mathrm{e}-1)$. Torng and Uthaisombut (1999) complete the analysis by providing matching lower bound instances.

- Kellerer *et al.* (1996) discuss $1 \,|\, r_j \,|\, \sum F_j$, the problem of minimising the total flow time of the jobs on a single machine under job release dates. By studying the preemptive relaxation, they derive a polynomial time approximation algorithm with worst-case performance guarantee $\mathrm{O}(\sqrt{n})$; here n denotes the number of jobs. Up to constant factors, the analysis is tight. Moreover, Kellerer *et al.* (1996) exhibit instances for which the integrality gap of the preemptive relaxation is $\Omega(\sqrt{n})$.

A difficult open problem is to analyse the preemptive relaxation for the open shop problem $O \,|\,|\, C_{\max}$. In an m-machine open shop, every job J_j consists of m operations $O_{1j}, \ldots, O_{mj}$ with processing times $p_{1j}, \ldots, p_{mj}$ that have to have processed on machines $M_1, \ldots, M_m$. The processing order of the m operations $O_{1j}, \ldots, O_{mj}$ is *not* prespecified; it may differ for different jobs, and it may be decided and fixed by the scheduler. At any moment in time, at most one operation from any job can be processed. The goal is to minimise the makespan. The preemptive version of the open shop is denoted by $O \,|\, pmtn \,|\, C_{\max}$. Whereas the non-preemptive problem $O \,|\,|\, C_{\max}$ is strongly NP-hard (Williamson *et al.*, 1997), the preemptive version $O \,|\, pmtn \,|\, C_{\max}$ can be solved in polynomial time (Gonzalez and Sahni, 1976).

A folklore conjecture states that the integrality gap of the preemptive relaxation for $O \,|\,|\, C_{\max}$ should be at most $3/2$. Settling this conjecture would mean a major breakthrough in the area. Here is an instance that demonstrates

that the integrality gap is at least $3/2$: consider an open shop with m machines and $n = m + 1$ jobs. For $j = 1, \ldots, m$ the job J_j consists of the operation O_{jj} with processing time $p_{jj} = m - 1$ on machine M_j, and with processing times 0 on the other $m - 1$ machines. The job J_{m+1} has m operations all with processing time 1. Then the optimal preemptive makespan is m, and the optimal non-preemptive makespan is $\lceil m/2 \rceil + m - 1$. As m becomes large, the ratio between non-preemptive and preemptive makespan tends to $3/2$.

Acknowledgments

I thank Martin Skutella for showing me the worst-case instance for preemptive makespan under job rejections.

References

Chekuri, C. and Motwani, R. (1999). Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, **98**:29–38.

Chekuri, C., Motwani, R., Natarajan, B. and Stein, C. (2001). Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, **31**:146–166.

Chudak, F. and Hochbaum, D. S. (1999). A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, **25**:199–204.

Dyer, M. E. and Wolsey, L. A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, **26**:255–270.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*. Freeman, New York.

Gonzalez, T. and Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the ACM*, **23**:665–679.

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

Graham, R. L. (1969). Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, **17**:416–429.

Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, **5**:287–326.

Hall, L. A., Schulz, A. S., Shmoys, D. B. and Wein, J. (1997). Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, **22**:513–544.

Hochbaum, D. S. and Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, **34**:144–162.

Hoogeveen, J. A., Lenstra, J. K. and Veltman, B. (1994). Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, **16**:129–137.

Hoogeveen, H., Skutella, M. and Woeginger, G. J. (2003). Preemptive scheduling with rejection. *Mathematical Programming*, **94**:361–374.

Kellerer, H., Tautenhahn, T. and Woeginger, G. J. (1996). Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96)*, pp. 418–426.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, Handbooks in Operations Research and Management Science 4, S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin (Eds.), North-Holland, Amsterdam, pp. 445–522.

Lenstra, J. K., Shmoys, D. B. and Tardos, É. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, **46**:259–271.

Munier, A. and König, J. C. (1997). A heuristic for a scheduling problem with communication delays. *Operations Research*, **45**:145–147.

Papadimitriou, C. H. and Yannakakis, M. (1990). Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, **19**:322–328.

Picouleau, C. (1992). Etude des problèmes d'optimisation dans les systèmes distribués. *Thèse*, l'Université Paris 6.

Potts, C. N. (1980). An algorithm for the single machine sequencing problem with precedence constraints. *Mathematical Programming Study*, **13**:78–87.

Torng, E. and Uthaisombut, P. (1999). A tight lower bound for the best-alpha algorithm. *Information Processing Letters*, **71**:17–22.

Veltman, B., Lageweg, B. J. and Lenstra. J. K. (1990). Multiprocessor scheduling with communication delays. *Parallel Computing*, **16**:173–182.

Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevastianov, S. V. and Shmoys, D. B. (1997). Short shop schedules. *Operations Research*, **45**:288–294.

Woeginger, G. J. (2000). A comment on scheduling on uniform machines under chain-type precedence constraints. *Operations Research Letters*, **26**:107–109.

ORDER SCHEDULING MODELS: AN OVERVIEW

Joseph Y-T. Leung and Haibing Li
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102, USA
{leung, hl27}@njit.edu

Michael Pinedo
Stern School of Business
New York University
New York, NY 10012, USA
mpinedo@stern.nyu.edu

Abstract Order scheduling models can be described as follows: A machine environment with a number of non-identical machines in parallel can produce a fixed variety of different products. Any one machine can process a given set of the different product types. If it can process only one type of product it is referred to as a dedicated machine, otherwise it is referred to as a flexible machine. A flexible machine may be subject to a setup when it switches from one product type to another product type. Each product type has certain specific processing requirements on the various machines. There are n customers, each one sending in one order. An order requests specific quantities of the various different products and has a release date as well as a due date (committed shipping date). After the processing of all the different products for an order has been completed, the order can be shipped to the customer. This paper is organised as follows. We first introduce a notation for this class of models. We then focus on various different conditions on the machine environment as well as on several objective functions, including the total weighted completion time, the maximum lateness, the number of orders shipped late, and so on. We present polynomial time algorithms for the easier problems, complexity proofs for NP-hard problems and worst case performance analyses as well as empirical analyses of heuristics.

Keywords: order scheduling, models, complexity results, heuristics.

1. INTRODUCTION

We consider a facility with m different machines in parallel. There are k different product types that can be produced on these m machines. Each product type $l = 1, 2, \ldots, k$ can be produced on a subset of the machines, namely $M_l \subseteq \{1, 2, \ldots, m\}$. To produce one unit of type l on a machine $i \in M_l$ requires a processing time t_{li}. When a machine $i \in \{1, 2, \ldots, m\}$ produces a batch of type l products, a setup time of s_{li} is required before the batch can be started. Assume there are n orders from n different clients. Order j requests a quantity $q_{lj} \geq 0$ of product type l, $j = 1, \ldots, n, \quad l = 1, \ldots, k$. For order j, the processing time required to produce items of type l on machine $i \in M_l$ is $p_{lij} = q_{lj} \cdot t_{li}$. Order j may have a release date r_j, a due date (delivery date) d_j, and a positive weight w_j. The completion time of order j, denoted by C_j, is the time when the last product for order j has been completed on one of the machines. Let C_{lj} denote the completion time of the type l production for order j on one of the machines. Clearly,

$$C_j = \max_l \{C_{lj}\}$$

The idea of measuring the overall completion time of an entire set of jobs (i.e. all the jobs required by one order) rather than the individual completion times of each part of any given order is somewhat new. There are several reasons for considering the orders rather than the individual jobs within the orders. First of all, shipping partial orders inevitably causes additional shipping cost. Secondly, it also causes extra management effort. Finally, some customers may require suppliers to ship complete orders. Therefore, suppliers have to wait until all products for an order are ready.

With regard to the completion times $C_1, \ldots, C_n$ of the n orders several objectives are of interest, namely, the makespan $C_{\max}$, the maximum lateness $L_{\max}$, the total weighted completion time $\sum w_j C_j$ of orders, the total weighted tardiness $\sum w_j T_j$ of orders, and the total weighted number of late orders $\sum w_j U_j$.

Clearly, the class of models described above is very rich. Several special cases are of interest, namely:

(i) *The fully dedicated case*: there are m machines and m product types; each machine can produce one and only one type.

(ii) *The fully flexible case*: the m machines are identical and each machine is capable of producing all k products.

(iii) *The arbitrary case*: there are no restrictions on the subsets M_l.

The classes of models have a large number of practical applications. Any Make-To-Order environment at a production facility with a number of flexible resources in parallel gives rise to models of the type described above.

Julien and Magazine (1990) presented two interesting applications of the models described above. The first example involves a manufacturer of computer peripherals such as terminals, keyboards and disk drives. Small businesses that purchase new computer systems order different quantities of each of these peripherals, and often require their entire order to be shipped together. From the manufacturer's point of view, it is advantageous to aggregate the demand of each peripheral and produce large batches in order to minimise the number of setups.

A second example that illustrates the models described is a pharmaceutical company that can produce different types of pills. Each type of pill needs to be bottled separately. However, for a given pill type, it is often necessary to have different bottle sizes. The pills and the bottles may be produced based on forecasts. However, the bottling and packaging stage is order driven. The customers, which may be drugstores or hospitals, order certain quantities of each product type (a product type being a bottle of a given size of a given pill type). The production setups are the switch-overs in the bottling facility.

Yang (1998) presented another example in the form of a car repair shop. Suppose each car has several broken parts that need to be fixed. Each broken part can only be fixed by a certain set of mechanics in the shop. Several mechanics can work on different parts of the same car at the same time. The car will leave the shop when every broken part is fixed.

In manufacturing systems that consist of two stages, different types of components (or subassemblies) are produced first in a pre-assembly stage, and then put together into final products (jobs) in an assembly stage. The pre-assembly stage consists of parallel machines (called feeding machines), each of which produces its own subset of components. Each assembly operation cannot start its processing until all the necessary components are fed in. As shown in several papers (Duenyas, 1994; Lee *et al.*, 1993; Potts *et al.*, 1995), there are many examples of such two-stage assembly systems. An interesting example arises in parallel computing systems, in which several programs (or tasks) are first processed independently on certain processors, and then gathered at a main processor for final data-processing. The main processor can only start its processing after all the programs have fed in their results. As noted by Sung and Yoon (1998), our models only deal with the pre-assembly stage in such two-stage systems.

The general problem has not received much attention in the literature. Some of the concepts underlying the general problem were introduced by Julien and Magazine (1990), who were probably the first to identify this type of problem as order scheduling. The problem was studied in more detail in a dissertation by Yang (1998).

Various special cases of the general problem have been considered in the literature. However, most of these special cases are significantly less compli-

cated than the general problem. An important special case is the case in which each order requests just a single product type. If, in addition, the machines are fully flexible and no product type requires any setup, then the problem immediately reduces to the standard parallel machine environment which is usually referred to as $Pm \mid \beta \mid \gamma$ in the literature. If the machines are not fully flexible and there is no setup time for any product type, the problem reduces to a more general parallel machine environment that is often referred to as *unrelated* machines in parallel ($Rm \mid \beta \mid \gamma$). There is a very rich literature on these parallel machine scheduling problems. For an overview, see Brucker (1995) and Pinedo (2002).

We propose the following notation for our class of scheduling problems. Our notation is an extension of the $\alpha \mid \beta \mid \gamma$ notation introduced by Graham *et al.* (1979). In what follows, we always assume that there are m machines in parallel and n orders that come in from n different customers. The fully dedicated case of this parallel machine environment is denoted by PDm, the fully flexible case by PFm, and the arbitrary case by PAm; when the m is omitted we assume that the number of machines is arbitrary. In the β field we include πk to refer to the fact that we have k different product types; the absence of the k indicates that the number of different product types may be arbitrary. In addition, we include in the β field an s when the setup times for all product types are identical, an s_l when the setup times for the various product types are different but identical for each machine, and an s_{li} when the setup times are dependent on both product types and machines. The absence of s_{li}, s_l, and s indicates that there is no setup time for any product type. Note that setup times do not make sense for the fully dedicated case. In addition, if all machines are identical, then $s_{li} = s_l$ for each machine $i = 1, 2, \ldots, m$. As an example of the notation, $PF6 \mid prmt, s, \pi 3 \mid L_{max}$ refers to the fully flexible case with six machines in parallel and three different product types. Each product type has the same setup time s, order j has a due date d_j, and preemptions are allowed. The objective is the minimisation of maximum lateness.

In the next section, we consider the fully dedicated case. The fully flexible case is considered in Section 3. Finally, we draw some conclusions in the last section.

2. THE FULLY DEDICATED CASE

As mentioned before, in the fully dedicated case, there are m machines and the number of product types is equal to m; each machine can produce only one type. Since machine i is the only machine that can produce type i and type i is the only type that can be produced on machine i, the subscript i refers to a machine as well as to a product type. We note that Wagneur and Sriskandarajah (1993) referred to this model as "open shops with job overlaps", while Ng *et*

al. (2003) called this model "concurrent open shops". This case is considerably easier than the other cases because there is no freedom in the assignment of jobs to machines. Each machine can start processing at time 0 and keeps on producing as long as there is a demand. The main issue here is the assignment of finished products to customers. For dedicated machines, the setup times do not play a role in scheduling, and can therefore be dropped from consideration.

For unweighted objectives, the following structural properties can be shown easily.

Lemma 1 *(i) The makespan $C_{\max}$ is independent of the schedule, provided that the machines are always kept busy whenever there are orders available for processing (i.e. provided unforced idleness is not allowed).*

(ii) If all $r_j = 0$ and $f_j(C_j)$ is increasing in C_j for all j, then there exists an optimal schedule for the objective function $f_{\max}$ as well as an optimal schedule for the objective function $\sum f_j(C_j)$ in which all machines process the orders in the same sequence.

(iii) If for some machine i there exists a machine h such that $p_{ij} \leq p_{hj}$ for $j = 1, \ldots, n$, then machine i does not play any role in determining the optimal schedule and may be ignored.

Some remarks with regard to these properties are in order. The second property does not hold for the more general problem in which the function $f_j(C_j)$ is not monotonic (e.g., problems that are subject to earliness and tardiness penalties). The third property is useful for reducing the size of an instance of a problem.

Consider the problem $PD \mid \beta \mid \sum f_j(C_j)$. Since this problem is strongly NP-hard, it is advantageous to develop dominance conditions or elimination criteria. We can prove the following order dominance result.

Lemma 2 *If in the problem $PD \mid\mid \sum f_j(C_j)$ there are two orders j and k such that $p_{ij} \leq p_{ik}$ for each $i = 1, 2, \ldots, m$, and*

$$\frac{\mathrm{d}f_j(t)}{\mathrm{d}t} \geq \frac{\mathrm{d}f_k(t)}{\mathrm{d}t}$$

then there exists an optimal schedule in which order j precedes order k.

Let us consider the total weighted completion time objective, $\sum w_j C_j$. Sung and Yoon (1998) showed the following result.

Theorem 3 *The problem $PD2 \mid\mid \sum w_j C_j$ is strongly NP-hard.*

Wagneur and Sriskandarajah (1993) considered the $\sum C_j$ objective. They presented a proof claiming that $PD2 \mid\mid \sum C_j$ is strongly NP-hard. Unfortunately, as pointed out by Leung *et al.* (2005), their proof is not correct. The

complexity status of this two machine problem remains open so far. However, Leung *et al.* (2002a) obtained the following result for three machines.

Theorem 4 *The problem* $PD3 \mid\mid \sum C_j$ *is NP-hard in the strong sense.*

Since the problem to minimise $\sum C_j$ is strongly NP-Hard with three or more machines, it makes sense to develop and analyse heuristics for this problem. A number of researchers have focused their attention on the development of heuristics and the following heuristics have been proposed for $PD \mid\mid \sum C_j$.

Definition 1 The Shortest Total Processing Time first (STPT) heuristic generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest total amount of processing over all m machines.

Definition 2 The Shortest Maximum Processing Time first (SMPT) heuristic generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest maximum amount of processing on any one of the m machines.

Definition 3 The Smallest Maximum Completion Time first (SMCT) heuristic first sequences the orders in nondecreasing order of p_{ij} on each machine $i = 1, 2, \ldots, m$, then computes the completion time for order j as $C_j' = \max_{i=1}^{m}\{C_{ij}\}$, and finally schedules the orders in nondecreasing order of C_j'.

Definition 4 The Shortest Processing Time first on the machine with the largest current load (SPTL) is a heuristic that generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest processing time on the machine that currently has the largest load.

Definition 5 The Earliest Completion Time first (ECT) heuristic generates a sequence of orders one at a time; each time it selects as the next order the one that would be completed the earliest.

The STPT and the SMPT heuristics have been studied by Sung and Yoon (1998). They focused on two machines. Wang and Cheng (2003) studied the m machine case. Besides the STPT and the SMPT heuristics, they also analysed the SMCT heuristic. The last two heuristics, i.e. SPTL and ECT, were proposed by Leung *et al.* (2002a).

It turns out that all these heuristics may perform quite poorly in their worst case. For example, Wang and Cheng (2003) obtained the following worst-case bound.

Theorem 5 *For the problem* $PDm \mid\mid \sum C_j$,

$$\frac{\sum C_j(H)}{\sum C_j(OPT)} \leq m$$

where $H \in \{STPT, SMPT, SMCT\}$.

Leung *et al.* (2002a) showed that SPTL is unbounded. They also obtained the following result.

Theorem 6 *For the problem* $PDm \mid\mid \sum C_j$,

$$\frac{\sum C_j(ECT)}{\sum C_j(OPT)} \leq m$$

It is not clear whether the worst-case bound is tight for these heuristics. For two machines, Leung *et al.* (2002a) presented an instance for which the ratio is 1.618 for the STPT heuristic. They also gave an instance for which the ratio is $\sqrt{2}$ for both ECT and SMCT.

Leung *et al.* (2002a) performed an extensive empirical analysis showing that among the five heuristics described above, the ECT rule performs on the average clearly the best.

For minimising $\sum w_j C_j$, the performance bounds of the weighted version of STPT, SMPT, SMCT remain unchanged (Wang and Cheng, 2003). Leung *et al.* (2003a) also modified the ECT and SPTL heuristics to take the weights of the orders into account. The new heuristics are referred to as the WECT and WSPL rules. In detail, the WECT heuristic selects the next order j^* which satisfies

$$j^* = \arg\min_{j \in \Omega} \left\{ \frac{C_j - C_k}{w_j} \right\}$$

where C_k is the completion time of the order that was scheduled immediately before order j^*. A postprocessing procedure interchanges order j^* with order k in case $C_{j^*} \leq C_k$ in order to obtain a better (at least no worse) solution. Note that the case $C_{j^*} \leq C_k$ occurs only when $p_{i^*j^*} = 0$, where i^* is the machine on which order k has, over all machines, the largest finish time. Assume that after the swap the order immediately before order j^* is order l. If $C_{j^*} \leq C_l$, we proceed with an interchange of order j^* with order l. We continue with this postprocessing until C_{j^*} is larger than the completion time of the order that immediately precedes it. Note that after each swap, the finish time of j^* either decreases or remains unchanged, while the finish time of each order that is swapped with j^* remains unchanged. This is due to the fact that order j^* has zero processing time on the machine on which the swapped order has its largest finish time. Thus, the postprocessing, if any, produces a solution that is no worse than the one without postprocessing. Note that following the WECT heuristic, there may at times be ties. Since ties may be broken arbitrarily, the WECT heuristic may lead to various different schedules. For the performance of WECT, we can show the following result.

Theorem 7 *For the problem* $PDm \mid\mid \sum w_j C_j$,

$$\frac{\sum C_j(WECT)}{\sum C_j(OPT)} \leq m$$

The proof of the above theorem turns out to be much more complicated than that of Theorem 6. With additional constraints of processing times for each order, we can show that the performance bounds of the heuristics can be much better (close to 2 or 3). For details, see Leung *et al.* (2003a).

We also note that Wang and Cheng (2003) proposed an approximation algorithm which has a worst-case ratio of $16/3$. The algorithm is based on a linear programming relaxation which is formulated on the time intervals geometrically divided over the time horizon. Leung *et al.* (2003a) also presented a 2-approximation algorithm which is based on a linear programming formulation on the completion time of the orders. However, the implementation of this algorithm is quite complicated.

Leung *et al.* (2003a) did an extensive experimental analysis of the five simple heuristics listed above as well as of the $16/3$-approximation algorithm. Experimental results show that among the five simple heuristics, WECT is the best. For instances with certain characteristics, the WECT rule performs even better than the $16/3$-approximation algorithm.

The NP-hardness of problem $PD1 \mid \pi 1 \mid \sum T_j$ is a direct consequence of the NP-hardness of $1 \mid\mid \sum T_j$, see Du and Leung (1990).

The maximum lateness $L_{\max}$ can be minimised by the *Earliest Due Date* rule; i.e., the next finished product is assigned to the customer with the earliest due date. Through an adjacent pairwise interchange argument the following theorem can be shown.

Theorem 8 *The Earliest Due Date rule solves the problem* $PD \mid\mid L_{\max}$, *and the Preemptive Earliest Due Date rule solves the problem* $PD \mid prmt, r_j \mid L_{\max}$.

In a more general setting, the processing of the orders are subject to precedence constraints and the objective function is the more general $f_{\max}$. Lawler (1973) developed an algorithm based on (backwards) dynamic programming that solves the single machine version of this problem in polynomial time. Lawler's algorithm can be extended in such a way that it generates optimal solutions for the more general PD machine environment.

Theorem 9 *The problem* $PD \mid prec \mid f_{\max}$ *can be solved in* $O(n^2)$ *time.*

The problem $PD1 \mid r_j, \pi 1 \mid L_{\max}$ is NP-hard, which is a direct consequence of the NP-hardness of $1 \mid r_j \mid L_{\max}$.

The total number of late jobs objective, $\sum w_j U_j$, is also of interest. When $w_j = 1$, Wagneur and Sriskandarajah (1993) showed the following result.

Theorem 10 *The problem* $PD2 \mid\mid \sum U_j$ *is NP-hard in the ordinary sense.*

In fact, Cheng and Wang (1999) showed that there exists a pseudo-polynomial time algorithm for every fixed $m \geq 2$. When the number of machines is arbitrary, Ng *et al.* (2003) showed the following result.

Theorem 11 *The problem* $PD \mid\mid \sum U_j$ *is NP-hard in the strong sense. The NP-hardness in the strong sense remains in effect even for the very restricted case* $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$.

For the problem $PD \mid d_j = d \mid \sum U_j$, Leung *et al.* (2002b) showed that the problem reduces to the Multiset Multicover (MSMC) problem (Rajagopalan and Vazirani, 1998). Thus, any algorithm solving the MSMC problem also solves the problem $PD \mid d_j = d \mid \sum U_j$. Leung, Li and Pinedo (2002b) adapted Rajagopalan and Vazirani's greedy algorithm for MSMC in such a way that it is also applicable to the $PD \mid d_j = d \mid \sum U_j$ problem. In the next theorem this modified version of the Rajagopalan and Vazirani algorithm is denoted by H_g. Leung *et al.* (2002b) obtained the following result for the H_g algorithm.

Theorem 12 *For* $PD \mid d_j = d \mid \sum U_j$, *if all* p_{ij} *and* d *are integers, then*

$$\frac{\sum U_j(H_g)}{\sum U_j(OPT)} \leq \mathcal{H}\left(\max_{1 \leq j \leq n}\left\{\sum_{i=1}^{m} \frac{p_{ij}}{c_i}\right\}\right)$$

where $c_i = GCD(p_{i1}, p_{i2}, \ldots, p_{in}, d)$ *for* $i = 1, 2, \ldots, m$, *and* $\mathcal{H}(k) \equiv \sum_{i=1}^{k}(1/i)$ *is the harmonic series. In addition, the bound is tight.*

It should be noted that Rajagopalan and Vazirani's greedy algorithm was designed for the weighted MSMC problem. If in our problem each order has a weight w_j, then it is also easy to adapt the greedy algorithm to solve $PD \mid d_j = d \mid \sum w_j U_j$. The approximation ratio of the revised greedy algorithm for $PD \mid d_j = d \mid \sum w_j U_j$ remains unchanged. Another observation for $PD \mid d_j = d \mid \sum w_j U_j$ is that it is in fact the dual problem of the multidimensional 0–1 knapsack problem (MKP) with an arbitrary number of dimensions. Thus, the resolution methods for MKP also shed light on solving $PD \mid d_j = d \mid \sum w_j U_j$. For a very recent survey for the MKP problem, the reader is referred to Fréville (2004).

For the restricted case $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$, Ng *et al.* (2003) proposed a $(d+1)$-approximation algorithm based on a linear programming relaxation. However, if we apply the greedy algorithm for $PD \mid d_j = d \mid \sum U_j$ to solve $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$, the approximation ratio is at most $\mathcal{H}(m)$. Thus, if $m \leq e^d$, the approximation ratio of the greedy algorithm would be better than that of the LP-based heuristic. In fact, by our

reduction, the $PD \mid p_{ij} = 0 \text{ or } 1, d_j = d \mid \sum U_j$ problem turns out to be a set multicover (SMC) problem, since the elements in each constructed set are unique but each element requires to be covered multiple times. Thus, any approximation algorithm for the SMC problem can be applied to solve $PD \mid p_{ij} = 0 \text{ or } 1, d_j = d \mid \sum U_j$ with the approximation ratio being preserved. Hochbaum (1996) presented several LP-based ρ-approximation algorithms for weighted SMC, where

$$\rho = \max_{1 \le j \le n} \left\{ \sum_{i=1}^{m} p_{ij} \right\}$$

Clearly, these ρ-approximation algorithms can be applied to solve $PD \mid p_{ij} = 0 \text{ or } 1, d_j = d \mid \sum w_j U_j$. Since $\mathcal{H}(\rho) < \rho$ for $\rho \ge 2$, it is easy to see that the approximation ratio of our greedy algorithm is still better.

Heuristics for $PD \mid\mid \sum U_j$ can be designed based on the ideas of the Hodgson–Moore (1968) algorithm which solves the single machine version of this problem, i.e. $PD1 \mid \pi 1 \mid \sum U_j$, to optimality. The Hodgson–Moore algorithm for the single machine version generates a schedule by inserting the jobs in a forward manner one at a time according to the EDD rule. Whenever a job is completed after its due date, the procedure selects among the jobs that are currently still part of the schedule the longest one and takes it out of the schedule. Once a job has been taken out of the schedule, it never can get back in. Using the main idea behind the Hodgson–Moore algorithm the following heuristic can be designed for $PD \mid\mid \sum U_j$. The orders are put in the schedule S in a forward manner one at a time; whenever an order j' that is put into the schedule is completed after its due date, one of the orders that are currently part of the schedule has to be taken out.

The selection of the order that has to be taken out can be done based on a priority ranking system. In order to make sure that not more than one order has to be deleted from the schedule, it pays to keep a set of candidate orders S_c with the property that the removal of any one order in S_c from S ensures that the rest of the orders in S are completed before their due dates.

First of all, the tardy order j' itself is already a candidate order, since all the orders that precede j' in S can be completed before their due dates. For each order $j \in S, j \neq j'$, if its removal from S enables order j' to be completed in time, then j becomes a candidate in S_c, otherwise, j will not become a candidate. It is clear that $1 \le |S_c| \le |S|$.

Secondly, for each candidate order $j \in S_c$, a weighted sum of all its processing times p_{ij} on the m machines, denoted by $W(S_c)_j$, has to be computed. The weight of machine i, denoted by w_i, is a function of the current load on machine i, denoted by CL_i, and the future workload of machine i due to all the orders that still have to be considered, denoted by FL_i. A typical weight

function can be

$$w_i = \omega_1 CL_i + \omega_2 FL_i$$

where ω_1 and ω_2 are the weights for CL_i and FL_i, respectively, for any $i, 1 \leq i \leq m$. With w_i, the weighted sum of each candidate order $j \in S_c$ is computed as

$$W(S_c)_j = \sum_{i=1}^{m} w_i p_{ij}$$

Finally, the candidate order to be taken out is the one with the maximum weighted sum, i.e. order j^* such that

$$W(S_c)_{j^*} = \max_{j \in S_c} \Big(W(S_c)_j \Big)$$

Leung *et al.* (2002b) have done an extensive empirical analysis of the heuristic above. They also proposed an exact algorithm that uses constraint propagation, backtracking, and bounding techniques. Their result shows that the exact algorithm can solve instances of moderate size in a reasonable running time; and the results of the above heuristic are quite close to those of the exact algorithm. Recently, Leung *et al.* (2005) generalised the heuristic above so that it is applicable to the weighted case.

3. THE FULLY FLEXIBLE CASE

In the fully flexible case, the m machines are identical and each machine is capable of producing all k products. Two sets of problems are of interest, namely

(i) the fully flexible cases without setup times,

(ii) the fully flexible cases with arbitrary setup times.

Clearly, the fully flexible case is more difficult than the fully dedicated case, the reason being that we have to take care of two issues for this case: besides sequencing the orders, we need also to assign the product types to the machines. Recall that for the fully dedicated case, we need only to consider the issue of sequencing the orders.

3.1 The Fully Flexible Case Without Setup Times

The problem $PF1 \mid \pi k, \beta \mid \gamma$ is identical to the problem $1 \mid \beta \mid \gamma$. In the following we will consider $m \geq 2$ only. As we shall see, there are similarities as well as differences between the case $PFm \mid \pi k, \beta \mid \gamma$ and the standard parallel machine environment $Pm \mid \beta \mid \gamma$.

For the objectives of $C_{\max}$, $L_{\max}$ and $\sum w_j T_j$, the complexities follow closely those of the standard parallel machine scheduling environment. Thus,

$$PF \mid prmt, \pi k \mid C_{\max} \quad \text{and} \quad PF \mid prmt, \pi k \mid L_{\max}$$

are solvable in polynomial time, whereas

$$PF2 \mid \pi 1 \mid C_{\max}, \; PF2 \mid \pi 1 \mid L_{\max} \quad \text{and} \quad PF2 \mid \pi 1 \mid \sum T_j$$

are NP-hard (since $PF1 \mid \pi 1 \mid \sum T_j$ is NP-hard).

On the other hand, the complexities are different for the $\sum C_j$ objective. It is well known that $P \mid\mid \sum C_j$ and $P \mid prmt \mid \sum C_j$ can be solved by the Shortest Processing Time first (SPT) rule (Pinedo, 2002). The following result was obtained by Blocher and Chhajed (1996).

Theorem 13 *The problem $PF2 \mid \pi k \mid \sum C_j$ is NP-hard in the ordinary sense.*

When $k = 2$, Yang (1998) showed that the problem remains NP-hard.

Theorem 14 *The problem $PF2 \mid \pi 2 \mid \sum C_j$ is NP-hard in the ordinary sense.*

But it is not known whether or not there exists a pseudo-polynomial time algorithm for the above ordinary NP-hard problem. When the number of machines is arbitrary, Blocher and Chhajed (1996) showed the following result.

Theorem 15 *The problem $PF \mid \pi k \mid \sum C_j$ is NP-hard in the strong sense.*

Consider now the preemptive version of this same problem. One can think of two different sets of assumptions for a preemptive version of this problem.

Under the first set of assumptions, the processing of a particular product type for a given order may be shared by multiple machines and the various machines are allowed to do this processing simultaneously. It is easy to show that the class of problems in this case, i.e. $PF \mid prmt, \beta \mid \gamma$ is identical to the class of problems $1 \mid prmt, \beta \mid \gamma$. This equivalence implies that this particular preemptive version is very easy.

Under the second set of assumptions for a preemptive version of this problem, any product for any given order can be processed partly on one machine and partly on an other. However, now we assume that if a product type for a given order is done on more than one machine, then these different processing times are not allowed to overlap. Leung *et al.* (2002c) showed that this particular preemptive version is hard.

Theorem 16 *The problem* $PF2 \mid prmt, \pi 2 \mid \sum C_j$ *is NP-hard in the ordinary sense.*

It is possible to design for $PF \mid \pi k \mid \sum C_j$ heuristics that have two phases. The first phase determines the sequence of the orders, and the second phase assigns the different products of an order to the machines. Based on these ideas, Blocher and Chhajed (1996) developed two classes of heuristics.

The first class of heuristics can be referred to as the *static two phase heuristics*. In these two phase heuristics, the orders are sequenced first, and then the different products for each order are assigned to the machines. Rules for sequencing the orders include:

- The *smallest average processing time first* (SAPT) rule sequences the orders in increasing order of $\sum_{l=1}^{k} p_{lj}/m$.
- The *smallest maximum completion time first* (SMCT) rule sequences the orders in increasing order of $C_{LPT}^{(j)}, j = 1, 2, \ldots, n$, where $C_{LPT}^{(j)}$ is the makespan of the schedule that is obtained by scheduling the different product types of order j on the m parallel machines according to the *longest processing time first* (LPT) rule, assuming each machine is available from time zero on.

After the sequence of orders is determined by one of the above rules, the product types of each order can be assigned to machines following one of the two assignment rules below:

- The *Longest Processing Time first* rule (LPT) assigns in each iteration an unassigned product type with the longest processing time to a machine with the smallest workload, until all product types are scheduled.
- The *Bin Packing* rule (BIN) starts by determining the completion time of an order using the LPT assignment rule above (just as a trial, not being the real assignment). This completion time is used as a target completion time (bin size). In each iteration, the BIN rule assigns an unassigned product type with the longest processing time to one of the machines with the largest workload. If the workload of the machine exceeds the target completion time after the assignment, then undo this assignment and try the assignment on the machine with the second largest workload. This procedure is repeated until the product type can be assigned to a machine without exceeding the target completion time. If assigning the product type to the machine with the smallest workload still exceeds the target completion time, then assign it to this machine, and reset the target completion time to the completion time of the product type on this machine.

Combinations of the sequencing rules with the assignment rules lead to four different heuristics: namely, SAPT-LPT, SAPT-BIN, SMCT-LPT, and SMCT-BIN.

The second class of heuristics may be referred to as the *dynamic two-phase heuristics*. In these heuristics, the sequence of orders is not fixed prior to the assignment of product types to machines, i.e., the sequence is determined dynamically. The heuristics still use the LPT rule or the BIN rule for the assignment. However, to determine the next order to be sequenced, a greedy approach is applied to make a trial assignment of the product types of all remaining orders by using either the LPT or the BIN rule, and the order that gives the smallest completion time is selected as the next order in the sequence. These two heuristics may be referred to as *Greedy-LPT* and *Greedy-BIN*.

Blocher and Chhajed (1996) did not obtain performance bounds for these heuristics. However, they did an experimental analysis and found that the results obtained with the heuristics are very close to the lower bound they developed. They also found that not one of the heuristics consistently dominates any one of the others. It turns out that these heuristics do have certain performance bounds. For details, the reader is referred to Leung *et al.* (2003b).

3.2 The Fully Flexible Case With Setup Times

The scheduling problems become considerably harder with arbitrary setup times, even for a single machine. Leung *et al.* (2003b) considered $PF1 \mid s_l \mid L_{\max}$ and $PF1 \mid s_l \mid \sum U_j$ and proved the following theorem.

Theorem 17 *The problems* $PF1 \mid s_l \mid L_{\max}$ *and* $PF1 \mid s_l \mid \sum U_j$ *are both NP-hard in the ordinary sense.*

This result is rather surprising, since $1 \mid\mid L_{\max}$ can be solved by the Earliest Due Date rule and $1 \mid\mid \sum U_j$ can be solved by the Hodgson–Moore algorithm.

Minimising $C_{\max}$ on one machine is solvable in polynomial time. All we need to do is to batch all requests of the same product type together and sequence the product types in an arbitrary order. In this way, each product type will incur its setup time at most once and $C_{\max}$ will be minimised. Unfortunately, Leung *et al.* (2003b) showed that we lose polynomiality when we have two or more machines.

Theorem 18 *The problems* $PF2 \mid s_l \mid C_{\max}$ *and* $PF2 \mid prmt, s_l \mid C_{\max}$ *are both NP-hard in the ordinary sense.*

For the objective of minimising the total completion time, some work has been done recently for the single machine case. Ng *et al.* (2002) showed the following result.

Theorem 19 *The problem* $PF1 \mid s, \pi k, p_{lj} \in \{0,1\} \mid \sum C_j$ *is strongly NP-hard.*

Interestingly, they mentioned that the complexity of $PF1 \mid s_l, \pi k, p_{lj} > 0 \mid \sum C_j$ remains an open problem. If there is a requirement for all the operations of any given product type to be scheduled contiguously (i.e. the operations for each product type must be scheduled in one batch), then the process is said to follow the *group technology* (GT) approach (see Gerodimos *et al.*, 1999; Ng *et al.*, 2002). While Gerodimos *et al.* (1999) showed that $PF1 \mid s_l, \pi k, GT, p_{lj} > 0 \mid \sum C_j$ is polynomially solvable, Ng *et al.* (2002) obtained the following complexity result.

Theorem 20 *The problem* $PF1 \mid s, \pi k, GT, p_{lj} \in \{0,1\} \mid \sum C_j$ *is strongly NP-hard.*

For two machines, we can derive the following result from Theorem 16.

Theorem 21 *The problem* $PF2 \mid prmt, s, \pi k \mid \sum C_j$ *is NP-hard in the ordinary sense.*

Consider now the case in which the orders have different release dates. Leung *et al.* (2003b) showed that no online algorithm (i.e. algorithms that operate without any knowledge of future order arrivals) can do as well as an optimal offline algorithm.

Theorem 22 *There is no optimal online algorithm for* $PF1 \mid r_j, s, \pi k \mid C_{\max}$ *and* $PF1 \mid prmt, r_j, s, \pi k \mid C_{\max}$.

4. CONCLUDING REMARKS

Order scheduling models have many real world applications. It is a relatively new idea to optimise the objective functions of the completion times of orders rather than individual completion time of the jobs included in the orders. Some work has been done in the past for special cases. Our goal has been to classify the previous work in a single framework based on the characteristics of machine environment and objectives. While the complexity of some of the problems are known, many remain open. It will be interesting to settle these issues in the future. Developing heuristics with provably good worst case bounds and/or good empirical performance is also a worthwhile direction to pursue.

References

Blocher, J. and Chhajed, D. (1996) The customer order lead-time problem on parallel machines. *Naval Research Logistics*, **43**:629–654.

Brucker, P. (1995) *Scheduling Algorithms*. Springer, Berlin.

Cheng, T. and Wang, G. (1999) Customer order scheduling on multiple facilities. *Technical Report* 11/98-9, Faculty of Business and Information Systems, The Hong Kong Polytechnic University.

Du, J. and Leung, J. Y.-T. (1990) Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, **15**:483–495.

Duenyas, I. (1994) Estimating the throughput of a cyclic assembly system. *International Journal of Production Research*, **32**:403–1410.

Fréville, A. (2004) The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, **155**:1–21.

Gerodimos, A., Potts, C., and Tautenhahn, T. (1999) Scheduling multi-operation jobs on a single machine. *Annals of Operations Research*, **92**:87–105.

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979) Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, **5**:287–326.

Hochbaum, D. (1996) Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum (Ed.), PWS Publishing Company, Boston, MA, pp. 94–143.

Julien, F. and Magazine, M. (1990) Scheduling customer orders—an alternative production scheduling approach. *Journal of Manufacturing and Operations Management*, **3**:177–199.

Lawler, E. (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, **19**:544–546.

Lee, C., Cheng, T., and Lin, B. (1993) Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, **39**:616–625.

Leung, J. Y.-T., Li, H. and Pinedo, M. (2002a) Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling*, accepted for publication.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2002b) Scheduling orders for multiple product types with due date related objectives. *European Journal of Operational Research*. Accepted for publication.

Leung, J. Y.-T., Lee, C. Y., Young, G. H. and Ng, C. W. (2002c) Minimizing total flow time in generalized task systems. Submitted.

Leung, J. Y.-T., Li, H., Pinedo, M. and Sriskandarajah, C. (2005) Open shops with jobs overlap—revisited. *European Journal of Operational Research*. Accepted for publication.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2003b) Order scheduling in a flexible environment with parallel resources. *Working Paper*.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2003c) Scheduling multiple product types with weighted objectives. *Working Paper*.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2003a) Scheduling orders for multiple product types to minimize total weighted completion time. *Working Paper*.

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

Ng, C., Cheng, T., and Yuan, J. (2002) Strong NP-hardness of the single machine multi-operation jobs total completion time scheduling problem. *Information Processing Letters*, **82**:187–191.

Ng, C., Cheng, T., and Yuan, J. (2003) Concurrent open shop scheduling to minimize the weighted number of tardy jobs. *Journal of Scheduling*, **6**:405–412.

Pinedo, M. (2002) *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, NJ.

Potts, C., Sevast'janov, S., Strusevich, V., Wassenhove, L., and Zwaneveld, C. (1995) The two-stage assembly scheduling problem: complexity and approximation. *Operations Research*, **43**:346–355.

Rajagopalan, S. and Vazirani, V. (1998) Primal–dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, **28**(2):525–540.

Sung, C. and Yoon, S. (1998) Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. *International Journal of Production Economics*, **54**:247–255.

Wagneur, E. and Sriskandarajah, C. (1993) Open shops with jobs overlap. *European Journal of Operational Research*, **71**:366–378.

Wang, G. and Cheng, T. (2003) Customer order scheduling to minimize total weighted completion time. In *Proceedings of the 1st Multidisciplinary Conference on Scheduling Theory and Applications*, pp. 409–416.

Yang, J. (1998) Scheduling with batch objectives. *Ph.D. Thesis*, Industrial and Systems Engineering Graduate Program, The Ohio State University, Columbus, OH.

Multi-criteria Scheduling

SCHEDULING IN SOFTWARE DEVELOPMENT USING MULTIOBJECTIVE EVOLUTIONARY ALGORITHMS

Thomas Hanne and Stefan Nickel
Fraunhofer Institute for Industrial Mathematics (ITWM)
Department of Optimisation
Gottlieb-Daimler-Str. 49
67663 Kaiserslautern
Germany
{hanne, nickel}@itwm.fhg.de

Abstract We consider the problem of planning inspections and other tasks within a software development (SD) project with respect to the objectives quality (no. of defects), project duration, and costs. The considered model of SD processes comprises the phases of coding, inspection, test, and rework and includes assigning tasks to persons and generating a project schedule. Based on this model we discuss a multiobjective optimisation problem. For solving the problem (i.e., finding an approximation of the efficient set) we develop a multiobjective evolutionary algorithm. Details of the algorithm are discussed as well as results of its application to sample problems.

Keywords: multiple objective programming, project management and scheduling, software development, evolutionary algorithms, efficient set.

1. INTRODUCTION

Today, the software industry is facing increasing requirements for the quality of its products. At the same time, controlling the costs and keeping the deadlines of software development (SD) projects have become increasingly important issues. Up to now there have been few formal approaches in practical use for supporting project managers, e.g., by the simulation and optimisation of SD processes, presumably because of uncertainties in these processes and complexities in the causal relationships. Within the last couple of years, simulation models have been developed for improving the understanding and planning of SD processes. Mostly, these models (for a survey see Kellner *et al.*, 1999) are based on a system dynamics approach, which is particularly useful

for modelling feedback loops on a macro-level but hardly supports a detailed representation of the considered system. For that purpose, a discrete-event simulation approach is better suited and allows for an individual representation of objects such as persons and items produced in an SD process (design documents, source code modules, etc).

Within two larger research projects, we developed a discrete-event simulation model for supporting project managers in the software industry (Neu *et al.*, 2002; Neu *et al.*, 2003; Münch *et al.*, 2002). It is planned to use that model either as a direct decision support tool for software project managers or within the context of consulting. Training applications, e.g. for students in computer science, are also envisioned. In our model, the assignment of tasks to persons (e.g., items to be coded, items to be inspected) is done arbitrarily, thus on a first-come–first-served (FCFS) basis. This means that items are treated in a given arbitrary order and the developers being next available become their authors. The assignment of coding and other SD tasks to people is relevant since the duration of the project, i.e. the makespan, depends on this assignment and, in general, the time required for performing a task depends on a person's productivity which is, for instance, influenced by their experience.

Our model focuses on the planning of inspections as a key technology for finding defects and, thus, for ensuring the quality of a software product (see Basili and Boehm, 2001). Questions to be answered are, for example, whether inspections should be applied, which documents should be inspected and by how many persons. The relationships between the inspection team size and the inspection technique on the one hand and the inspection effectiveness on the other hand has been analysed during many experiments and in real-life settings (see e.g. Briand *et al.*, 1997). It is generally assumed that finding defects (at an early stage) in inspections is more effective (i.e. lower costs per found defect) than at a later stage, i.e. during testing, and that some inspection techniques seem to be superior to others. More differentiated results in combination with the planning of a specific SD project are, however, not available. As well as inspections, coding activities, rework, and testing are also represented in the model. Other activities such as those in the requirement and design phases are neglected for the moment.

For the model, it is assumed that a source code item of a length known in advance is coded first. Then it is to be inspected and, after that, subject to rework. After that, an item is going to be tested and is then again subject to rework. During coding, defects are produced, some of which are detected during inspection and testing, and removed during rework. For doing so, a person has to be assigned to an item as its author who performs the coding. For the inspection, several persons have to be assigned as inspectors. These persons must not be the same as the author. Also for the testing, one person (not the same as the author) has to be assigned. The reworking of a document

is done again by its author. Details on the organisation of such a process can be found in Ebenau and Strauss (1995) and Humphrey (1989).

Because of the unsatisfactory FCFS task assignment and the missing possibility of optimising other planning parameters within the discrete-event simulation model, we have reformulated the SD model as a combined assignment and parameter optimisation problem that can be solved separately (i.e. without connection to the original discrete-event model) by some tools for optimisation.

Here we discuss an evolutionary algorithm (EA) approach for planning inspections and scheduling staff. Evolutionary algorithms such as genetic algorithms (Holland, 1975) or evolution strategies (Schwefel, 1981; Bäck *et al.*, 1991) have been shown to be robust approaches applicable to a wide range of optimisation problems. Applications in the area of scheduling can be found in Wang and Uzsoy (2002), Herrmann and Lee (1995), Herrmann *et al.* (1995), Della Croce *et al.* (1995) and Yun (2002).

For the last 15 years, evolutionary algorithms have also increasingly been developed and used for multiobjective optimisation problems. Surveys on such multiobjective evolutionary algorithms (MOEAs) are given in the papers by Fonseca and Fleming (1995) Horn (1997) and Tamaki *et al.* (1996), and in the monographs by Coello Coello *et al.* (2002), Deb (2002) and and Zitzler (1999). More recent research results are included in the proceedings of the first and the second International Conferences on Evolutionary Multi-Criterion Optimisation (Zitzler *et al.*, 2001; Fonseca *et al.*, 2003). Other multiobjective metaheuristics such as simulated annealing (see Czyżak and Jaszkiewicz, 1998) and genetic local search (see Ishibuchi and Murata, 1998) have also led to competitive results for multiobjective optimisation problems but are not discussed here for lack of space. For surveys, comparative studies and further bibliography on various multiobjective metaheuristics, see Hansen (1998), Jaszkiewicz (2001) and Ehrgott and Gandibleux (2002).

Results on applying MOEAs to specific scheduling problems can be found in Celano *et al.* (1999), Cochran *et al.* (2003) or Esquivel *et al.* (2002). General results on multicriteria scheduling problems are presented in T'kindt and Billaut (2002). A survey on using metaheuristics for multiobjective scheduling is provided by Landa Silva and Burke (2002).

This paper is organised as follows. In Section 2 we present our model for the planning of inspections and the scheduling of staff. In Section 3, the MOEA for solving the corresponding multiobjective optimisation problem is outlined. Section 4 presents results of applying the MOEA to the multiobjective SD planning problem. The paper ends with some conclusions and an outlook to future work.

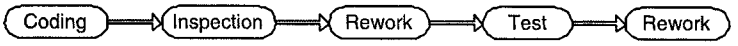


Figure 1. Sequence of tasks in software development.

2. A MULTIOBJECTIVE SOFTWARE INSPECTION PROBLEM

The basic assumption of our software process model is that a given number n of source code items has to be produced. The size of each item and a measure of its complexity are known in advance.[1] The coding tasks are done by a team of m developers such that each item is coded by one person called its author. After coding, an item may be subject to an inspection. This means that one or several persons from the team read the item and try to find defects. After the inspection, the item is subject to some rework for eliminating the found defects. After that, a document may be subject to some testing (and subsequent rework), which also serves the purpose of finding defects. The sequence of tasks for each item is represented in Figure 1. We assume all these tasks to be done by a team of m persons, each of them being able to perform any task.

Unlike the other tasks, inspections are assumed to be done in a preempting way such that persons involved in coding or testing may interrupt their tasks and read the documents to be inspected in between. This is necessary because inspections in practice require some kind of synchronisation of the inspectors (e.g., a common meeting). This means that inspections of the same document (unlike other tasks) must be scheduled in parallel. Inspections themselves are assumed to be non-preemptive. Similar problems in scheduling with priorities specific to the task occur in real-time operating systems (see e.g. Lamie, 1997, and Mäki-Turja *et al.*, 1999).

2.1 Assigning Tasks (Decision Variables)

According to our general assumptions each item has to be coded by exactly one person. Therefore, we consider the assignment of persons to coding tasks by an n-dimensional vector *author* with

$$author_i \in \{1, \ldots, m\} \tag{2.1}$$

for $i \in \{1, \ldots, n\}$.

For the inspection team size, $no\_inspectors_i$, we assume

$$no\_inspectors_i \in \{0, \ldots, no\_inspectors_{max}\} \text{ for all } i \in \{1, \ldots, n\} \tag{2.2}$$

[1]This and similar simplifications are necessary in order to make the problem treatable. In subsequent research it is planned to analyse a stochastic version of this problem which should allow a better representation of the situations in reality.

with a 0-component indicating that the corresponding item is not subject to inspection and considering a maximum team size $no\_inspectors_{max} \leq m - 1$.

For $i \in \{1, \ldots, n\}, k \in \{1, \ldots, no\_inspectors_i\}$ let

$$inspector_{ik} \in \{1, \ldots, m\} \tag{2.3}$$

indicate which persons are assigned to the inspection of item i. The restrictions

$$inspector_{ik} \neq author_i \text{ for all } i, k, \tag{2.4}$$

indicate that the author of a document is not allowed to be an inspector of the same document and, of course, the inspectors must be pairwise different, i.e.

$$inspector_{ik} \neq inspector_{il} \text{ for } k \neq l \text{ and } i \in \{1, \ldots, n\}. \tag{2.5}$$

Similarly, persons are assigned for testing an item, i.e.

$$tester_i \in \{1, \ldots, m\} \tag{2.6}$$

with

$$tester_i \neq author_i \text{ for all } i \in \{1, \ldots, n\}. \tag{2.7}$$

While for the other activities the duration genuinely depends on item and person attributes (see below) allowing them to be endogenously determined within the model, the test time is given exogenously as a decision variable. The reason is that more and more testing increases the number of defects found (although with a decreasing rate). Thus, test times are determined, for instance, by using a pre-specified test intensity, ti, used as a multiplicative factor, i.e.

$$tt_i = ti \cdot cplx_i \cdot size_i, \tag{2.8}$$

thus defining the test times as proportional to the size of an item weighted by its complexity. In reality it may be the case that the test time is not pre-specified but depends on the number of defects found per time unit: for example, with the motivation to reach a desired quality of the source code. This case is considered in the simulation model but will be neglected here for simplicity.

For determining the temporal sequence for scheduling tasks, we use priority values for the coding and testing activities as further decision variables of the model,

$$priority\_c_i \in [0, 1], \tag{2.9}$$

$$priority\_t_i \in [0, 1], \tag{2.10}$$

for $i \in \{1, \ldots, n\}$. A higher priority value indicates that the corresponding task should be scheduled before a task with a lower value, if possible. For the inspections such priority information is not required since we assume them to

be done directly after the coding of the item (allowing the inspectors to interrupt other current activities). In general, we assume that developers perform all their coding tasks within a project before starting the first testing activities.

In the original discrete-event simulation model constructed with the simulation tool Extend, the above decision variables are set implicitly by some kind of FCFS logic. This means that at the start of a simulation run, items are batched with persons according to a given order (determined by item and person numbers) as long as persons are available. Later on, items waiting for their next processing steps are batched with the next person becoming available. While determining an FCFS solution, for all the above decision variables, values consistent with that solution are determined. Within our optimisation model such an FCFS solution can be reproduced and serves as a benchmark solution for judging the quality of solutions generated by the MOEA.

2.2 Working Times and Effectiveness of Work

For the processing times, we assume that these depend on the jobs to be processed, i.e. the sizes of the items, their complexities and their domains, and the programmers assigned, especially their experiences, skills, and other human factors. Since not all of these factors can be modelled explicitly (especially because of a lack of data) we consider only skills as determining factors for the individual productivities. For each developer $k \in \{1, \ldots, m\}$, skill values are considered for the different activities, i.e. coding (coding productivity skill, cps_k) and inspection (inspection productivity skill, ips_k). The skills are assumed to be measured on the $[0, 1]$ interval. Multiplied by given values for maximum productivities (corresponding to skill values of 1), i.e. a maximum coding productivity, mcp, and a maximum inspection productivity, mip, the productivities of developers for the various tasks can be determined. Multiplied by the size of an item, $size_i$, and considering an adjusting factor for the complexity of the item, $cplx_i$, the specific working times can be calculated as follows (in the case of variables specific to coding, test, and rework we omit for clarity the indexes specific to the unique persons assigned to these tasks):

Coding times:

$$ct_i = size_i \cdot cplx_i/(mcp \cdot cps_k) \tag{2.11}$$

Inspection times:

$$it_{ik} = size_i \cdot cplx_i/(mip \cdot ips_k) \tag{2.12}$$

The rework times depend on the number of defects to be reworked and on the skills of the person doing the rework. Since the rework activities are closely connected to the coding activities, the same skill values are also used for determining the rework times, rt_i. This is done by using a factor expressing

the relationship between rework productivity (number of defects per hour) and coding productivity (number of lines of code per hour), and the average defect size, ads:

$$rt_i := fd_i \cdot cplx_i/(ads \cdot mcp \cdot cps_k) \tag{2.13}$$

In this way, we calculate the times for rework after inspection, rt_i^1, and after testing, rt_i^2, depending on the found defects fd_i^1 and fd_i^2, respectively.

As for the working times, the quality of work measured by the number of defects produced, found, and removed during the various activities depends on attributes of an item, i.e. its size and complexity, and attributes of the person assigned to the task. As for the times, we assume specific skill attributes for the persons measuring the quality of work on a $[0, 1]$-scale. The number of defects produced when an item i is coded by author k is assumed to be

$$pd_i = size_i \cdot cplx_i \cdot mdd/cqs_k \tag{2.14}$$

where cqs_k is the coding quality skill of k and mdd denotes the minimum defect density, i.e. the defect density of a highly skilled developer.

For an individual inspector k, it is assumed that the probability of finding a defect is a result of his or her defect detection skill, $dds_k \in [0, 1]$, multiplied by a factor, *itf*, expressing the effectiveness of the inspection technique and the influence of other organisational factors. Thus, inspector k's individual probability of overlooking a defect is $1 - itf \cdot dds_k$. For the inspection team, the effect of double counting found defects is considered by using the team probability of overlooking a defect (which is the product of the individual probabilities) for calculating the number of found defects:

$$fd_i^1 = pd_i \cdot \left(1 - \prod_{k=1}^{no\_inspectors_i} (1 - itf \cdot dds_k)\right) \tag{2.15}$$

For the rework, it is assumed that all found defects are removed but some of them not correctly, or that new defects are produced in a proportional relationship to the correctly removed ones:

$$rd_i^1 = rdf \cdot fd_i^1 \tag{2.17}$$

with *rdf* being a rework defects factor with $rdf < 1$. For the defects found during test, we assume that these are as follows in an exponential relationship to the testing time:

$$fd_i^2 = d_i \cdot (1 - e^{-dfr \cdot tqs_k \cdot tt_i}) \tag{2.18}$$

where d_i are the defects remaining after coding, inspection, and rework. tqs_k is the testing quality skill of k and *dfr* is a defect find rate. The rework after testing is done similarly as after inspection leading to new defects rd_i^2 which are calculated according to (2.17).

In the discrete-event simulation model, the skill values are dynamically varied, especially by employing a learning model. These effects are neglected in the simplified model used for optimisation.

In general, the result of an activity (e.g., the number of defects produced during coding or found during an inspection) as well as the time needed for performing it can be considered as random since many human effects cannot be predicted with a sufficiently high accuracy. In the discrete-event simulation model (see Neu *et al.*, 2002, 2003) these influences are represented by stochastic distributions. For the model considered here, however, we apply a deterministic approach based on expected values. It is easily possible to consider a "security span" in these deterministic values. A general discussion of the motivation and validity of the above formulas for working times and effectiveness of work is provided by Neu *et al.* (2002, 2003).

2.3 Objectives

From a practical point of view (see, e.g., Abdel-Hamid and Madnick, 1991), the following three objectives are frequently considered for software development processes:

(a) the quality of the product measured by the eventual overall number of defects of the documents produced during a project,

(b) the duration of the project (its makespan), and

(c) the costs or total effort of the project.

Based on the model above, the corresponding objective functions can be formalised as follows for the optimisation:

(a) The total number of defects, td, at the end of the project is a simple and effective measure for the quality of a software project. td is calculated by

$$td = \sum_i d_i. \tag{2.19}$$

where

$$d_i = pd_i - fd_i^1 + rd_i^1 - fd_i^2 + rd_i^2 \tag{2.20}$$

is the total number of defects in item i at the end of the process.

(b) Assuming that there are no specific dependencies among the coding tasks and that inspections tasks are done in an "interrupt fashion", waiting times do not occur prior to the testing of items.

Interrupts for the inspection of other author's documents are done in between where it is assumed that associated inspectors are immediately

available when an item is finished with coding. This assumption can be justified because inspection times are comparably small and, in practice, people usually have some alternative tasks for filling "waiting times".

For a person assigned to the testing of an item he or she has to wait until the item is finished with coding, inspection, and rework. An item can therefore not be tested until it is ready for testing and its tester is available. The specific times of each task are calculated by constructing a schedule for all tasks to be done, i.e. a complete schedule for the SD project which comprises several weeks or months. Based on this, the project duration, du, can simply be calculated by the maximum finishing time of the tasks.

(c) Project costs can be assumed to be proportional to the project effort which is the total time spent by the team members for accomplishing all tasks of the project:

$$tc = c \cdot \sum_i (ct_i + \sum_k it_{ik} + rt_i^1 + tt_i + rt_i^2) \tag{2.21}$$

where c are the unit costs of effort [EUR/h]. Thus, in the case of waiting times, we assume that the persons can do some tasks outside the considered project, i.e., that these times are not lost.

The considered multiobjective optimisation problem can then be formulated as

$$\text{"}min\text{"}(td(x), du(x), tc(x)) \tag{2.22}$$

for the decision variables

$$x = (author, no\_inspectors, inspector, tester, tt, priority\_c, priority\_t) \tag{2.23}$$

subject to the above constraints (2.1)–(2.21).

"min" means that each of the objectives should be minimised. Usually, the objectives can be considered to be conflicting such that there exists no solution that optimises all objectives simultaneously. As a solution in the mathematical sense, generally the set of efficient solutions is considered. An efficient solution is an alternatives for which there does not exist another one which is better in at least one objective without being weaker in any other objective, or formally: A multiobjective optimisation problem is defined by

$$\text{"}min\text{"}f(x) \tag{2.24}$$

with $x \in A$ (set of feasible solutions) and $f : R^n \rightarrow R^q$ being a vector-valued objective function. Using the Pareto relation " $\leq$ " defined by

$$x \leq y :<=> x_i \leq y_i \; \forall i \in \{1, \ldots, q\} \text{ and } x_i < y_i \; \exists i \in \{1, \ldots q\} \tag{2.25}$$

for all $x, y \in R^q$, the set of efficient (or Pareto-optimal) alternatives is defined by

$$E(A, f) := \{x \in A : \nexists y \in A : f(y) \leq f(x)\} \tag{2.26}$$

See Gal (1986) for more details on efficient sets.

When the efficient set is determined, or a number of (almost) efficient solutions is calculated, then further methods may be applied to elicit preference information from the decision maker (the project manager) and for calculating some kind of compromise solution (see Zeleny, 1982; Steuer, 1986; Vincke, 1992; Hanne, 2001a). For instance, some interactive decision support may be applied for that purpose. Below, a multicriteria approach based on the calculation of the (approximately) efficient set is presented.

3. A MULTIOBJECTIVE EVOLUTIONARY ALGORITHM FOR SCHEDULING SD JOBS

3.1 General Framework

In the following, we sketch a new MOEA suitable for approximating the efficient set of the considered multiobjective SD scheduling problem.

The basic idea of EAs is that from a set of intermediary solutions (population) a subsequent set of solutions is generated, inspired by concepts of natural evolution such as mutation, recombination, and selection. From a set of "parent solutions" (denoted by M^t) in generation $t \in N$, a set of "offspring solutions" (denoted by N^t) is generated using some kind of variation. From the offspring set (or the offspring set united with the parent set) a subsequent solution set (the parents of the next generation) is selected. Note that the described algorithm is based on the more technical treatment of MOEAs outlined in Hanne (1999, 2000, 2001b). Due to the distinguishing of a parent and an offspring population, a separate archive (as frequently used for MOEAs) is not required. Specific mechanisms for supporting the diversity of a population, such as fitness sharing, did not turn out to be necessary according to the numerical results with the problem and have, therefore, not been used.

Figure 2 illustrates the general framework of the evolutionary algorithm. Aspects of adapting an EA to the given problem, an approach of considering its multiobjective nature and other details are treated below.

3.2 Data Structures

One of the most important issues in applying evolutionary algorithms to specific types of problems is their problem-specific adaptation (Michalewicz, 1998). This concerns the choice of data types for representing instances of solutions to the given problem and tailoring the evolutionary operators to the data types and the considered class of problems. One of the main reasons for

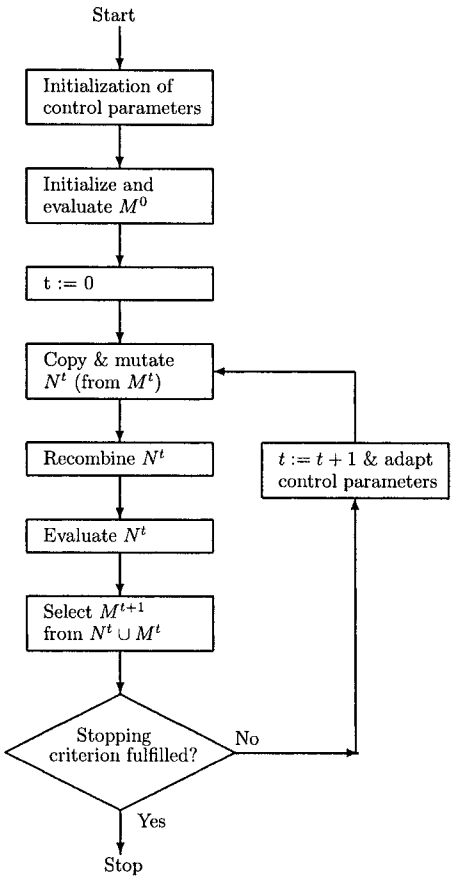


Figure 2. General framework of the MOEA.

this is that, in this way, the search space can be represented adequately without losing too much search effort and time in exploring "uninteresting" regions or for "repairing" infeasible solutions.

For the given problem, an essential amount of data is required to define a specific instance of the problem. This concerns the characterisation of items by their attributes, e.g. their sizes, complexities, and domains, the characterisation of persons by attributes such as their skills, and other global parameters relevant for a specific SD project, e.g. maximum inspection team sizes. A solution instance to the problem (i.e. a feasible alternative) is defined by those data specified in (2.23). Discrete variables are represented by integer numbers, real variables by floating-point numbers.

3.3 The Starting Population

For the starting population we require $\mu^0 \geq 1$ feasible solutions to the given project. In our implementation, we generate these solutions as random solutions. This is done by assigning a random author to each item, i.e.

$$author_i := uniform(1, \ldots, m) \text{ for } i \in \{1, \ldots, n\}, \tag{3.1}$$

where $uniform(1, \ldots, m)$ denotes a random number according to the discrete uniform distribution on $\{1, \ldots, m\}$.

The inspection team size for each item is set to a default value: for example, 3. For each item the inspectors and testers are assigned in the same fashion as the author but assuring that author and inspectors are not the same and that inspectors are mutually different for one item.

The priority values for tasks are assigned by

$$priority\_c_i := uniform[0, 1] \tag{3.2}$$

$$priority\_t_i := uniform[0, 1] \tag{3.3}$$

with $uniform[0, 1]$ denoting a random variable according to the continuous distribution on the interval $[0, 1]$.

In our MOEA software it is also possible to generate FCFS solutions (see above) similar to those used in the simulation model. Experiments have shown that these FCFS solutions are usually much better, especially with respect to the makespan, than the random solutions. Unfortunately, these solutions turned out to be bad as starting solutions for the MOEA, presumably because FCFS solutions constitute some kind of local optima which bound the algorithm's effectiveness. However, after just a small number of generations the MOEA started with random solutions as above has generated better solutions than FCFS solutions.

3.4 Mutation

For each item $i \in \{1, \ldots, n\}$ do with probability p_m the following reassignment: the persons responsible for each of the tasks related to that item are re-assigned according to the mechanism (3.1) used in creating a random starting solution.

The inspection team size is varied by mutations which are distributed around the previous inspection team size. This is done by normally distributed mutations similar to those used in evolution strategies, i.e.

$$no\_inspectors_i := no\_inspectors_j + round(normal(0, \sigma)) \tag{3.4}$$

where $normal(0, \sigma)$ denotes a random variable according to the normal distribution with expected value 0 and variance σ. For the new inspection team

size, the restriction (2.2) has to be observed. This is done by delimiting the value for $no\_inspectors_i$ at the interval borders 0 and $no\_inspectors_{max}$. So in the case of an increased inspection team size, additional inspectors have to be determined randomly.

All these re-assignments of decision variables are done ensuring that the restrictions (2.1)–(2.22) are kept. In the case of infeasible values, new random values are calculated.

3.5 Recombination

For performing the recombination of solutions, all offspring entities are randomly grouped into pairs. For each such pair of solutions, say a and b, perform a recombination of data with probability p_{reco1}. If a recombination is going to be performed, then for each item $i, i \in \{1, \ldots, n\}$, with a probability of p_{reco2} the assigned authors, inspectors, testers, testing times, and priorities of tasks are exchanged. This always leads to new feasible solutions.

3.6 Evaluation

In contrast to many other applications of evolutionary algorithms, the considered SD scheduling software requires complex procedures for evaluating an alternative (i.e. the calculation of the objective values) specified by data as in (2.23). While the calculation of the total number of defects, $td(x)$, and the total costs, $tc(x)$, is rather trivial for an alternative x, the calculation of the project duration, $du(x)$, requires the construction of a specific schedule. This proceeding necessitates the consideration of restrictions concerning the precedence of tasks (see Figure 1) and their preemption (see above).

The high-level procedure for calculating a schedule is as follows: All the tasks of one phase are planned before those of a subsequent phase. So first all coding tasks are scheduled. Then all inspections tasks are planned. It is assumed that inspections should take place as soon as possible after finishing the coding of an item. Since inspection tasks have a higher importance (requirement of some synchronisation) than other tasks it is assumed that they interrupt a coding activity of an inspector, which leads a prolongation of this task and a corresponding postponement of the subsequent coding tasks. Rework and test are scheduled after that without allowing interrupts of other tasks.

For the (non-preemptive) scheduling of the tasks of one type, there are two basic approaches implemented. Method 1 is based on observing the given assignment of persons to tasks and scheduling the tasks for one person according to the priorities, $priority\_c$ for coding and rework, $priority\_t$ for tests. Method 2 is based on the FCFS logic discussed above. This means that, ignoring pre-specified assignments and priorities, tasks are scheduled according in

the sequence of their first possible starting times being assigned to the earliest available person.

Computational experiments with these scheduling methods have indicated that the MOEA has difficulties in finding solutions being scheduled by method 1 that are better with respect to the project duration compared with a complete FCFS solution. While an FCFS solution works economically by avoiding waiting times, these are a significant problem in the case of pre-specified assignments and priorities. Here, the MOEA has difficulties in finding good solutions in the high-dimensional search space. On the other hand, a pure FCFS solution may lead to unnecessarily long project durations, for instance, because of starting time-demanding tasks too late. Trying out combinations of methods 1 and 2 (and other variants of these methods) has shown that using method 1 for scheduling coding and rework and method 2 for testing leads to superior results. So the comparatively long-lasting coding tasks at the beginning are scheduled by optimised assignments and priorities while the later testing tasks are scheduled in an FCFS fashion, which allows the filling of waiting time gaps. This concept of constructing a schedule has been employed for the numerical results described in Section 4.

3.7 Selection

The selection step of the MOEA is the only one that requires a special consideration of the multiobjective nature of the optimisation problem. For ordinary, scalar optimisation problems the objective value of each alternative is used for evaluating its fitness and for determining whether or in which way an alternative contributes to the subsequent generation. For instance, in canonical genetic algorithms, the probability for an alternative to reproduce is proportional to its fitness and in evolution strategies, only the best alternatives are selected as parents for the next generation (elitist selection).

For multiobjective optimisation problems, several criteria are to be considered for judging an alternative's fitness. A simple traditional approach to this problem is to find some mapping of the q criteria to a single one. This can, for instance, be done by weighting the objective values or applying a different kind of scalarisation to them. A common disadvantage to this approach is that it does not lead to a representative calculation of efficient solutions. Instead, solutions may concentrate in a specific region of the efficient set (genetic drift). In particular, in many cases not all efficient solutions are solutions to the scalar substitute problem (Fonseca and Fleming, 1993, 1995).

Therefore, various approaches have been developed for a more sophisticated multiobjective selection. One of them is the straightforward idea of just using information included in the dominance (Pareto) relation. For instance, the dominance grade (see Hanne, 1999, 2001b) of an alternative $a \in A$ is defined

by the number of alternatives which dominate it, i.e.

$$domgrade(a) := |\{b \in M^t \cup N^t : f(b) \geq f(a)\}| \tag{3.5}$$

If several alternatives have the same dominance grade, those from the parent population are preferred to ensure certain conservative properties (strong elite preservation) required for the convergence analysis of the MOEA (see Hanne, 1999). In particular, alternatives being efficient within the parent population may only be replaced by offspring alternatives which dominate them. This selection approach has been applied for the numerical experiments described below. It may, however, be the case that other selection rules not satisfying these properties show a faster progress towards the efficient frontier. These issues are still open questions in the research of MOEAs.

4. RESULTS

4.1 Test Problems and Settings of the MOEA

For performing numerical tests we have implemented the above MOEA and procedures related to the handling of SD projects within MATLAB. For testing the evolutionary algorithm, test problems have been generated choosing the following technical parameters with respect to problem size etc:

- number of items: $n = 100$
- number of staff members: $m = 20$
- maximum coding productivity: $mcp = 25$ [loc/h]
- minimum defect density: $mdd = 0.02$ [defects/loc]
- maximum inspection productivity: $mip = 175$ [loc/h]
- inspection technique factor: $itf = 0.45$
- test intensity: $ti = 0.05$
- defect find rate: $dfr = 0.1$
- rework defects factor $rdf = 0.1$
- average defect size: $ads = 8$ [loc]
- unit costs: $c = 150$ [EUR/h].

Item and person attributes were initialised with random values for defining a test problem instance. For the skill attributes, a normal distribution with an expected value 0.5 and a variance of 0.1 is assumed (but ensuring that the values are in $[0, 1]$). This means that the persons on the average reach 50%

of the optimum skill values. For the item size, a lognormal distribution with expected value 300 [loc] and variance 120 is applied. For the item complexity, a normal distribution with expected value 1 and variance 0.1 is assumed.

For the applied evolutionary algorithm, the following parameters have been chosen:

- no. of generations: $t_{max} = 1000$
- population size (no. of parent solutions): $\mu = 30$
- no. of offspring solutions: $\lambda = 30$
- strategy type = + (consider parents and offspring for selection)
- mutation probability: $p_{mut} = 0.15$
- recombination probabilities: $p_{reco1} = 0.45$
- $p_{reco2} = 0.15$.

The above probability values for the MOEA are determined by choosing the best parameters from a series of systematic experiments based on varying them over an interval in equidistant steps. Altogether, 74 runs of the MOEA have been performed for finding good parameter values. As a criterion for judging the performance of an MOEA the average relative improvement (compared with respect to the FCFS solution) for the maximum improvement (over the population) in the three objective values and for three stochastic repetitions of the MOEA run have been chosen. Because of running time reasons, for these experiments a smaller population size ($\mu = 10, \lambda = 10$) and a smaller number of generations $t_{max} = 20$) than above have been used. Another approach for determining parameter values for an EA based on the idea of applying another (meta) EA is outlined in Hanne (2001a). A more refined analysis of these parameter values did not seem to be necessary since their effect on the MOEA was not that high according to Table 1. Thus, the MOEA works rather robustly with respect to these parameters.

4.2 Numerical Results of the MOEA

In Figures 3–8 the basic results of applying an MOEA with the above properties to a given test problem are shown. The figures correspond to a single run of the algorithm but are typical for the algorithm applied to SD scheduling problems. In Figures 3–5, the results with respect to one of the three objective functions are shown. The distributions of respective objective values within the populations is visualised for some of the generations by using box plots. Especially, the box plots show the maximum, median, and minimum values. In Figures 6–8, for the populations of some selected generations the values for

Table 1. Some sorted results from the systematic variation of MOEA control parameters.

p_{mut}	p_{reco}	p_{reco2}	Avg. rel. improvement
0.15	0.45	0.15	8.30
0.25	0.05	0.05	8.25
0.15	0.25	0.45	8.16
0.15	0.15	0.35	7.94
0.05	0.25	0.05	7.82
0.15	0.35	0.15	7.76
0.05	0.45	0.35	7.76
0.25	0.35	0.45	7.53
0.25	0.35	0.35	7.36
0.05	0.45	0.25	7.32

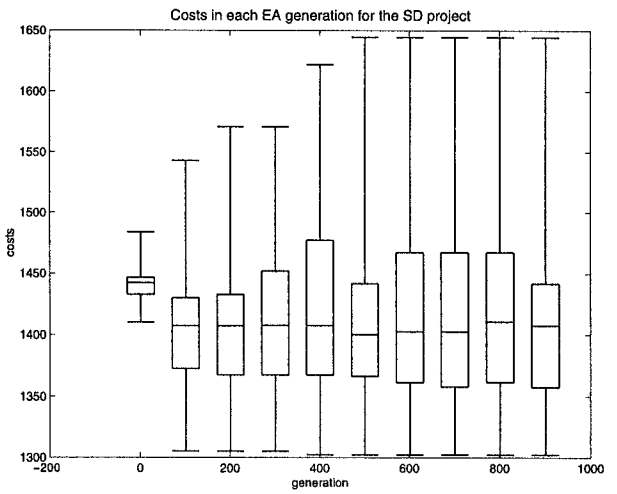


Figure 3. Costs of solutions depending on the generation no.

two objective functions are represented in the two-dimensional space for each combination of two objectives.

The box plots show that the most significant improvements with respect to the best values for the objectives costs and duration take place within the first 200 generations of the MOEA (Figures 3 and 4). Therefore, the corresponding two single-objective problems could be solved quite easily using an evolutionary optimisation approach. With respect to the number of defects, there are enhancements until about generation 500. But even after that generation, there is no steady state of the populations. For instance, worst and median objective

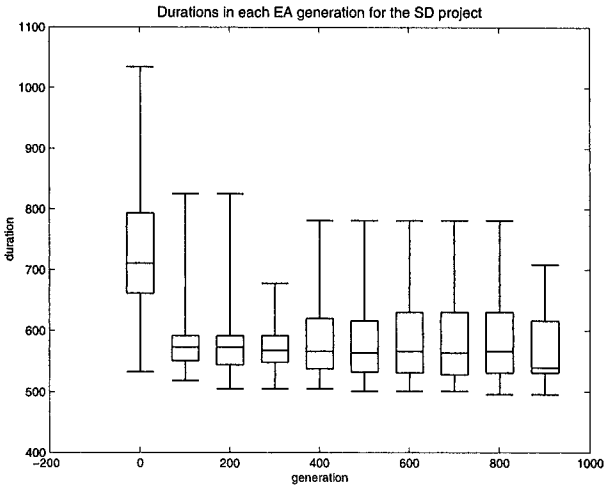


Figure 4. Duration (makespan) of solutions depending on the generation no.

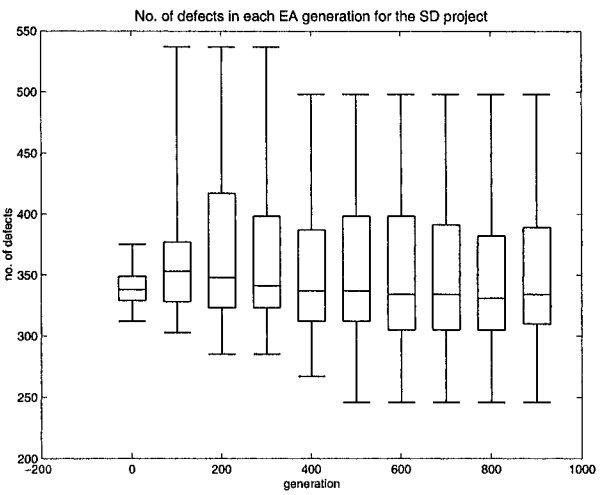


Figure 5. Defects of solutions depending on the generation no.

values for the duration can be improved without impairing the other objectives. In general, the bandwidth of objective values does not decrease beyond a specific level during the evolution. This reflects the fact that the objectives are conflicting. Therefore, an approximation of the efficient set consists of a

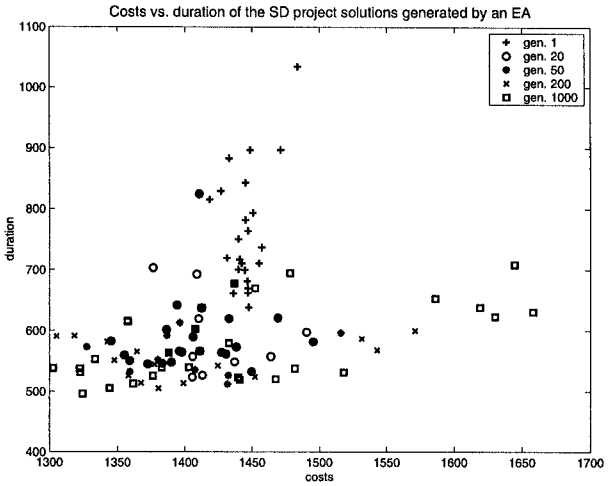


Figure 6. Costs and durations in solutions of several generations.

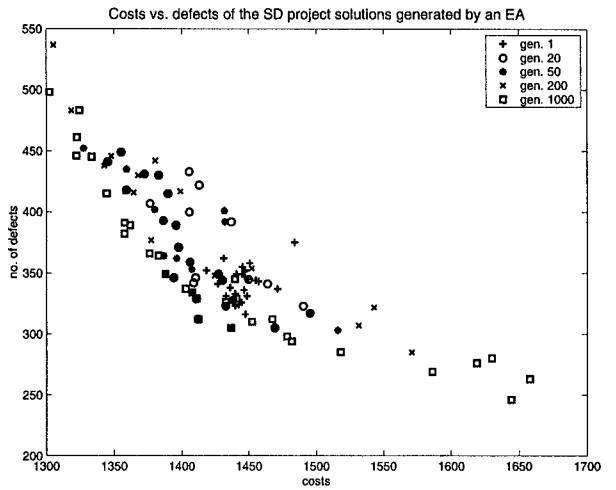


Figure 7. Costs and defects in solutions of several generations.

diverse set of solutions and requires a higher amount of computation than for optimising each single objective.

This result also becomes evident by considering the two-dimensional projections of the populations for several generations (Figures 6–8). As can be seen, there is a continuing progress towards the minimisation of the objectives

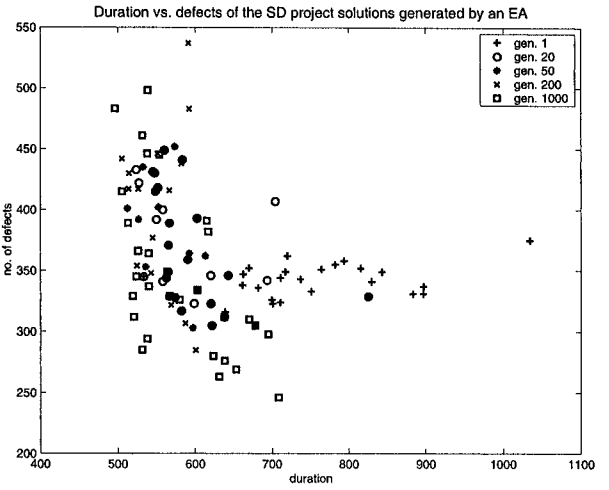


Figure 8. Durations and defects in solutions of several generations.

and the diversity of the solutions. For instance, considering the population of generation 1,000 in Figure 6, there are several solutions which are better with respect to the duration and the costs although there is no significant progress with respect to the best values in these objectives. Figures 7 and 8 show for generation 1,000 significant progress in the no. of defects (compared with generation 200) leading to a more diverse population. For some part, this is only possible by decreasing other objective values, e.g. by allowing higher costs. Altogether, a more diverse population provides a better basis for decision making. Of course, it is always possible (if desired) to decrease this scope: for example, by applying aspiration levels for some or all of the objectives. On the other hand, it is possible to get a denser representation of the efficient set by increasing the population size which, however, increases the computational effort.

Even for a population size of 30 and an evolution of 1000 generations, the computational requirements of the Matlab code are quite high. After performing some improvements of the algorithm by profiling it, the running times are still about six hours on an 850 MHz computer for a 1,000 generation MOEA with the above parameters. About 90% of the running time is required for evaluating the fitness function, i.e. calculating project schedules. An analysis of the Matlab code has shown that major parts of the running time are used for operations which can be significantly speeded up by using more specific data structures. Therefore, porting the code to C/C++ and replacing inefficient data structures will be an issue of our future work.

Table 2. Results for the MOEA compared with an FCFS solution.

	td	du	tc	Avg
Avg	7.13	6.87	10.87	8.29
Min	4.98	5.30	5.99	6.47
Max	9.15	8.27	20.36	11.04

In Table 2, results from 10 stochastic repetitions of the MOEA with a given problem instance are shown. The average, minimum, and maximum improvements (in percent) in each criterion, and for the unweighted average of all criteria are listed. The results indicate a sufficient robustness of the MOEA for producing improved solutions.

Further numerical experiments with other project sizes and varying heterogeneity of the material showed the following general results. With an increasing heterogeneity of a project, there is on the average more potential of optimisation which the MOEA can exploit. Small projects show a larger range of improvement for the MOEA with respect to the total number of defects. With respect to the project duration, the results are not that unique. For small and rather homogeneous projects, the FCFS scheduling delivers very good results which could, on the average, not be beaten by the MOEA. In contrast to that, the improvements for the MOEA are enormous for small but rather heterogeneous projects. For medium-size projects the improvement potentials of the MOEA are significant in both cases. For larger projects the improvement possibilities seem to decrease. With respect to the costs, the MOEA improvements are largest for medium-size projects with a large heterogeneity.

5. SUMMARY AND CONCLUSIONS

In this paper, we have presented a novel model of the software development process comprising the phases coding, inspection, test, and rework. For this model, originally developed as a discrete-event simulation model, there are various variables to be fixed corresponding to process parameters, task assignment, and scheduling. With respect to the usually important aspects of quality, costs, and projects duration, we have formulated the model as a multiobjective optimisation problem. For solving it, an evolutionary algorithm has been designed based in part on some additional scheduling heuristics. The application of the algorithm to test instances of the problem has shown significant improvements with respect to all of the objectives compared to a first-come first-served solution implicitly used within the original simulation model. The populations generated by the MOEA are broadly distributed with respect to the approx-

imation of the efficient set such that a decision maker may revert to clearly distinguishable alternatives for planning a software development project.

As yet, there have been no studies comparing our algorithm with other approaches. With respect to the pure MOEA a comparative study with other metaheuristics is in preparation. With respect to the specific scheduling problem, a comparison with other techniques is difficult because of the various problem-specific adaptations of the algorithm such as the usage of a priority-based subalgorithm for constructing schedules. Moreover, a comparison of the results of our algorithm to those of a "real scheduler" is impeded by the multiobjective nature of our problem and the combination with a parameter optimisation problem.

For the future development of our basic SD model, a number of extensions are planned. On the one hand, the behavioural model of the developers is going to be refined, especially by considering human effects such as learning or fatigue. On the other hand, we aim at a better consideration of the stochastic nature of SD processes. Also a re-scheduling approach is envisioned for project accompanying usage of the model. To some extent, these extensions are already considered in the discrete-event simulation model, though not in the optimisation model (in order to keep it simple). A stochastic model variant will be used for a further study which should serve for the analysis of the robustness of our optimisation method.

Up to now, the optimisation model has been used as a stand-alone tool. This will be changed by coupling it with the simulation software such that solutions calculated by the optimisation model may serve as solutions to be used for the simulation model. In this way, the inefficient FCFS solutions can be replaced by better ones.

Both the discrete-event simulation model as well as the optimisation model will be validated and/or adapted to a specific industrial setting by the usage of real-life data on processes in software development. It is also planned to equip the model with a user-friendly interface to facilitate its application into practice.

Acknowledgments

The authors gratefully acknowledge the support of this research given in connection with the SEV project and the ProSim project by the German Bundesministerium für Bildung und Forschung (SEV) and the Stiftung Rheinland-Pfalz für Innovation (ProSim, project no. 559). The authors also thank several anonymous referees for helpful comments on an earlier version of this paper.

References

Abdel-Hamid, T. and Madnick, S. E. (1991) *Software Project Dynamics. An Integrated Approach*, Prentice-Hall, Englewood Cliffs, NJ.

Bäck, T., Hoffmeister, F. and Schwefel H.-P. (1991) A survey of evolution strategies. In: *Genetic Algorithms, Proceedings of the 4th International Conference*, R. K. Belew, and L. B. Booker (Eds.), Morgan Kaufmann, San Mateo, CA, pp. 2–9.

Basili, V. and Boehm, B. (2001) Software defect reduction top 10 list. *Computer*, **34**:135–137.

Briand, L. C., Laitenberger, O. and Wieczorek, I. (1997) Building resource and quality management models for software inspections, *Technical Report*, International Software Engineering Research Network ISERN-97-06.

Celano, G., Fichera, S., Grasso, V., La Commare, U. and Perrone, G. (1999) An evolutionary approach to multi-objective scheduling of mixed model assembly lines. *Computers and Industrial Engineering*, **37**:69–73.

Cochran, J. K., Horng, S.-M. and Fowler, J. W. (2003) A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers and Operations Research*, **30**:1087–1102.

Coello Coello, C. A., Van Veldhuizen, D. A. and Lamont, G. B. (2002) *Evolutionary Algorithms for Solving Multi-objective Problems*, Kluwer, New York.

Czyżak, P. and Jaszkiewicz, A. (1998) Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, **7**:34–47.

Deb, K. (2002) *Multi-objective Optimization Using Evolutionary Algorithms* Wiley, Chichester.

Della Croce, F., Tadei, R. and Volta, G. (1995) A genetic algorithm for the job shop problem. *Computers and Operations Research*, **22**:15–24.

Ebenau, R. G. and Strauss, S. H. (1994) *Software Inspection Process*, McGraw-Hill, New York.

Ehrgott, M. and Gandibleux, X. (Eds.) (2002) *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Survey*. Kluwer, Boston, MA.

Esquivel, S., Ferrero, S., Gallard, R., Salto, C., Alfonso, H. and Schütz, M. (2002) Enhanced evolutionary algorithms for single and multiobjective optimization in the job shop scheduling problem. *Knowledge-Based Systems*, **15**:13–25.

Fonseca, C. M. and Fleming, P. J. (1993) Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Genetic Algorithms: Proceedings of the 5th International Conference*, S. Forrest (Ed.), Morgan Kaufmann, San Mateo, CA, pp. 416–423.

Fonseca, C. M. and Fleming, P. J. (1995) An overview of evolutionary algorithms in multiobjective optimization, *Evolutionary Computation* **3**:1–16.

Fonseca, C. M., Fleming, P. J., Zitzler, E., Deb, K. and Thiele, L. (Eds.) (2003) *Evolutionary Multi-Criterion Optimization. Proceedings of 2nd International Conference, EMO 2003*, Faro, Portugal, April 8–11, Springer, Berlin.

Gal, T. (1986) On efficient sets in vector maximum problems—A brief survey, *European Journal of Operations Research* **24**:253–264.

Hanne, T. (1999) On the convergence of multiobjective evolutionary algorithms, *European Journal of Operational Research* **117**:553–564.

Hanne, T. (2000) Global multiobjective optimization using evolutionary algorithms, *Journal on Heuristics* **6**:347–360.

Hanne, T. (2001a) *Intelligent Strategies for Meta Multiple Criteria Decision Making* Kluwer, Boston, MA.

Hanne, T. (2001b) Global multiobjective optimization with evolutionary algorithms: Selection mechanisms and mutation control. In *Evolutionary Multi-Criterion Optimization*, E. Zitzler, *et al.* (Eds.), Springer, Berlin, pp. 197–212.

Hansen, M. P. (1998) Metaheuristics for multiple objective combinatorial optimization, *Ph.D. Thesis*, IMM-PHS-1998-45, Technical University of Denmark, Lyngby.

Herrmann, J. W. and Lee, C.-Y. (1995) Solving a class scheduling problem with a genetic algorithm. *ORSA Journal of Computing* **7**:443–452.

Herrmann, J. W., Lee, C.-Y. and Hinchman, J. (1995) Global job shop scheduling with a genetic algorithm. *Production and Operations Management* **4**:30–45.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI.

Horn, J. (1997) Multicriterion decision making. In *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.), IOP Publishing and Oxford University Press, Bristol and New York, pp. F1.9:1–F1.9:15.

Humphrey, W. S. (1989) *Managing the Software Process*, Addison-Wesley, Reading, MA.

Ishibuchi, H. and Murata, T. (1998) Multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics* **28**:392–403.

Jaszkiewicz, A. (2001) Multiple objective metaheuristic algorithms for combinatorial optimization. *Habilitation Thesis*, Poznan University of Technology, Poznan.

Kellner, M. I., Madachy, R. J. and Raffo, D. M. (1999) Software process simulation modeling: Why? What? How? *Journal of Systems and Software* **46**:91–105.

Lamie, B. (1997) Preemption threshold. *Real-Time Magazine* **97**(3):46–47.

Landa Silva, J. D. and Burke, E. K. (2002) *A Tutorial on Multiobjective Metaheuristics for Scheduling and Timetabling*, paper presented at the 1st Workshop on Multiobjective Metaheuristics, Paris,

Mäki-Turja, J., Fohler, G. and Sandström, K. (1999) Towards efficient analysis of interrupts in real-time systems. In *Proceedings of Work-in-Progress Session, 11th EUROMICRO Conference on Real-Time Systems*, York.

Michalewicz, Z. (1998) *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edn, revised and extended. Springer, Berlin.

Münch, J., Berlage, T., Hanne, T., Neu, H., Nickel, S., von Stockum, S. and Wirsen, A. (2002) Simulation-based evaluation and improvement of SW development processes. *SEV Progress Report* No. 1.

Neu, H., Hanne, T., Münch, J., Nickel, S. and Wirsen, A. (2002) Simulation-based risk reduction for planning inspections. In *Product Focused Software Process Improvement*, M. Oivo, and S. Komi-Sirviö, (Eds.), Lecture Notes in Computer Science, Vol. 2559, Springer, Berlin, pp. 78–93.

Neu, H., Hanne, T., Münch, J., Nickel, S. and Wirsen, A. (2003) *Creating a Code Inspection Model for Simulation-Based Decision Support*. Paper presented at ProSim '03, Portland State University, May 3–4.

Schwefel, H.-P. (1981) *Numerical Optimization of Computer Models*, Wiley, Chichester.

Steuer, R. E. (1986) *Multiple Criteria Optimization: Theory, Computation, and Application*, Wiley, New York.

Tamaki, H., Kita, H. and Kobayashi, S. (1996) Multi-objective optimization by genetic algorithms: A review. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation*, T. Fukuda, and T. Furuhashi, (Eds.), IEEE Press, Piscataway, NJ, pp. 517–522.

T'kindt, V. and Billaut, J.-C. (2002) *Multicriteria Scheduling*, Springer, Berlin.

Vincke, P. (1992) *Multicriteria Decision-Aid*, Wiley, Chichester.

Wang, C.-S. and Uzsoy, R. (2002) A genetic algorithm to minimize maximum lateness on a batch processing machine, *Computers and Operations Research*, **29**:1621–1640.

Yun, Y. S. (2002) Genetic algorithm with fuzzy logic controller for preemptive and non-preemptive job-shop scheduling problems, *Computers and Industrial Engineering* **43**:623–644.

Zeleny, M. (1982) *Multiple Criteria Decision Making*, McGraw-Hill, New York.

Zitzler, E. (1999) *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, Swiss Federal Institute of Technology (ETH) Zurich TIK-Schriftenreihe Nr. 30, Diss ETH No. 13398, Shaker, Aachen.

Zitzler, E., *et al.*, (Eds.) (2001) *Evolutionary Multi-Criterion Optimization, Proceedings of 1st International Conference (EMO 2001)*, Zurich, Switzerland, March 2001, Springer, Berlin.

SCHEDULING UNIT EXECUTION TIME TASKS ON TWO PARALLEL MACHINES WITH THE CRITERIA OF MAKESPAN AND TOTAL COMPLETION TIME

Yakov Zinder and Van Ha Do
Department of Mathematical Sciences, University of Technology, Sydney, Australia
Yakov.Zinder@uts.edu.au and vhdo@it.uts.edu.au

Abstract Two extensions of the classical scheduling model with two parallel identical machines and a partially ordered set of unit execution time tasks are considered. It is well known that the Coffman–Graham algorithm constructs for this model a so-called ideal schedule: that is, a schedule which is optimal for both makespan and total completion time criteria simultaneously. The question of the existence of such a schedule for the extension of this model, where each task has a release time, has remained open over several decades. The paper gives a positive answer to this question and presents the corresponding polynomial-time algorithm. Another straightforward generalization of the considered classical model is obtained by the introduction of multiprocessor tasks. It is shown that, despite the fact that a slightly modified Coffman–Graham algorithm solves the makespan problem with multiprocessor tasks for arbitrary precedence constraints, generally an ideal schedule does not exist and the problem with the criterion of total completion time turns out to be NP-hard in the strong sense even for in-trees.

Keywords: scheduling, parallel machines, precedence constraints, multiprocessor tasks.

1. INTRODUCTION

This paper is concerned with two extensions of the classical scheduling model which can be stated as follows. Suppose that a set of n tasks (jobs, operations) $N = \{1, \ldots, n\}$ is to be processed by two parallel identical machines. The restrictions on the order in which tasks can be processed are given in the form of an anti-reflexive, anti-symmetric and transitive relation on the set of tasks and will be referred to as precedence constraints. If task i precedes task k, denoted $i \rightarrow k$, then the processing of i must be completed before the processing of k begins. If $i \rightarrow k$, then k is called a successor of i and i is called a predecessor of k. Each machine can process at most one task at a

time, and each task can be processed by any machine. If a machine starts to process a task, then it continues until its completion, i.e. no preemptions are allowed. The processing time of each task i is equal to one unit of time. Both machines become available at time $t = 0$. Since preemptions are not allowed and the machines are identical, to specify a schedule σ it suffices to define for each task i its completion time $C_i(\sigma)$. The goal is to find a schedule which minimizes some criterion $\gamma(\sigma)$.

In the standard three-field notation this problem is denoted by

$$P2 \,|\, prec, p_j = 1 \,|\, \gamma, \tag{1}$$

where $P2$ specifies that tasks are to be processed on two identical machines, $prec$ indicates presence of precedence constraints, and $p_j = 1$ reflects the fact that each task is a unit execution time (UET) task, i.e. requires one unit of processing time.

The criteria of makespan

$$C_{max}(\sigma) = \max_{1 \le i \le n} C_i(\sigma) \tag{2}$$

and total completion time

$$C_{\Sigma}(\sigma) = \sum_{1 \le i \le n} C_i(\sigma) \tag{3}$$

are among the most frequently used in scheduling theory. Several algorithms have been developed for the $P2 \,|\, prec, p_j = 1 \,|\, C_{max}$ problem. One of them, known as the Coffman–Graham algorithm (Coffman and Graham, 1972), also solves the $P2 \,|\, prec, p_j = 1 \,|\, C_{\Sigma}$ problem (see Lawler *et al.*, 1993; Coffman *et al.*, 2003). Following Coffman *et al.* (2003), we will say that a schedule is ideal if it is optimal for both (2) and (3) simultaneously.

A straightforward generalization of (1) is the $P2 \,|\, prec, r_j, p_j = 1 \,|\, \gamma$ problem, which differs from the former one only by the assumption that the processing of each task j cannot commence before the given integer release time r_j. Section 2 gives a positive answer to the open question of the existence of an ideal schedule for the model with release times. A polynomial-time algorithm, presented in Section 2, combines the ideas of the Coffman–Graham algorithm and the Garey–Johnson algorithm (Garey and Johnson, 1977), which solves the $P2 \,|\, prec, r_j, p_j = 1 \,|\, L_{max}$ problem with the criterion of maximum lateness, where

$$L_{\max}(\sigma) = \max_{1 \le i \le n} [C_i(\sigma) - d_i]$$

and d_i is a due date associated with task i. A relevant result can also be found in Zinder (1986), which is concerned with the problem of finding a schedule

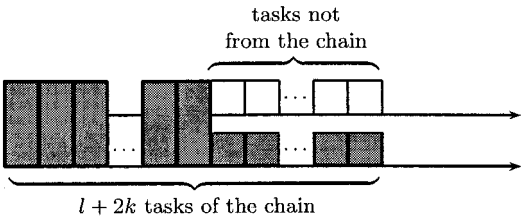


Figure 1. Schedule σ_1.

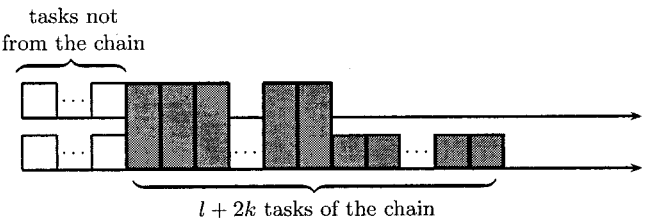


Figure 2. Schedule σ_2.

with the smallest value of C_Σ among all schedules optimal for the criterion of maximum lateness.

Another way of generalizing (1) is the introduction of multiprocessor tasks. The new problem, denoted by $P2\,|\,prec, p_j = 1, size_j\,|\,\gamma$ differs from the original one only by the assumption that each task requires for processing either one or two machines simultaneously, which is indicated by the parameter $size_j$. If $size_j = 1$, then task j needs only one machine. If $size_j = 2$, then task j can be processed only if two machines are used simultaneously. An arbitrary instance of the model with multiprocessor tasks does not necessarily allow an ideal schedule. Indeed, for arbitrary positive integers k and l, where $l > k$, consider the set of $4k + l$ tasks such that $l + 2k$ tasks form a chain and all remaining tasks do not precede or succeed any other tasks. The first l tasks in this chain require two machines simultaneously and all other tasks can be processed on one machine. It is easy to see that in any schedule, optimal for the criterion C_{max}, each task which does not belong to the chain must be processed in parallel with one of the last $2k$ tasks from the chain. Such a schedule, say σ_1, is depicted in Figure 1, where shaded tasks are the tasks constituting the chain.

Figure 2 depicts another schedule, say σ_2, where all tasks which do not belong to the chain are processed during the first k time units, followed by the tasks constituting the chain.

It is easy to see that σ_2 is not optimal for the criterion C_{max} and that

$$C_\Sigma(\sigma_1) - C_\Sigma(\sigma_2) = k(l-k) > 0$$

Section 3 shows that the $P2 \,|\, prec, p_j = 1, size_j \,|\, C_\Sigma$ problem is NP-hard in the strong sense even if the precedence constraints are restricted to in-trees. This result strengthens the result in Brucker *et al.* (2000), where NP-hardness has been proven for series-parallel graphs, and complements Zinder *et al.* (2002), where NP-hardness has been proven for out-trees.

2. IDEAL SCHEDULES WITH RELEASE TIMES

2.1 Algorithms

The model considered in this section assumes that each task j is assigned a non-negative integer r_j (referred to as a release time) and that processing of this task cannot commence before time r_j. Without loss of generality it will be assumed that the relation $i \rightarrow j$ implies the inequality $r_i \leq r_j - 1$ and that $\min_{i \in N} r_i = 0$.

The algorithm presented in this section uses the ideas introduced in Coffman and Graham (1972) and Garey and Johnson (1977). The central concept of Garey and Johnson (1977) is the notion of consistent due dates. In the process of constructing an ideal schedule, consistent due dates will be calculated for different subsets of the set N. Let $J \subseteq \{1, \ldots, n\}$. Let $\{D_i : i \in J\}$ be a set of arbitrary positive integers associated with tasks constituting J. We will refer to such integers as due dates. For any task $u \in J$ and any two numbers s and d such that

$$r_u \leq s \leq D_u \leq d, \tag{4}$$

$S(u, s, d, J)$ will denote the set of all tasks $k \in J$ such that $k \neq u$, $D_k \leq d$, and either $u \rightarrow k$ or $r_k \geq s$. Similar to Garey and Johnson (1977), we will say that the set of due dates $\{D_i : i \in J\}$ is consistent or that the due dates are consistent if $r_u \leq D_u - 1$, for all $u \in J$, and for every task $u \in J$ and any two integers s and d satisfying (4), either $|S(u, s, d, J)| = 2(d - s)$ and $D_u = s$, or $|S(u, s, d, J)| < 2(d - s)$.

Garey and Johnson (1977) developed a procedure which for any given set of positive due dates $\{d_i : i \in J\}$ either determines a set of consistent due dates $\{D_i : i \in J\}$ satisfying the inequalities $D_i \leq d_i$, for all $i \in J$, or determines that such consistent due dates do not exist at all. If due dates $\{d_i : i \in J\}$ are consistent, then the procedure returns these due dates as the set $\{D_i : i \in J\}$. The complexity of this procedure is $O(|J|^3)$. Some important results from Garey and Johnson (1977) are presented below in the form of three theorems. In what follows, the expression "schedule α for the set J" means that only tasks belonging to J are considered with precedence constraints which exist

between these tasks in the original problem. In other words, such α schedules only tasks from J and ignores the existence of other tasks.

Theorem 1 *Let $\{d_i : i \in J\}$ be a set of arbitrary positive due dates. If the procedure from Garey and Johnson (1977) fails to determine the set of consistent due dates, then there is no schedule α for the set J satisfying the inequalities $C_i(\alpha) \leq d_i$, for all $i \in J$.*

Theorem 2 *Let $\{d_i : i \in J\}$ be a set of arbitrary positive due dates and let $\{D_i : i \in J\}$ be the set of the corresponding consistent due dates calculated by the procedure from Garey and Johnson (1977). Then a schedule α for the set J satisfies the inequalities $C_i(\alpha) \leq d_i$, for all $i \in J$, if and only if it satisfies the inequalities $C_i(\alpha) \leq D_i$, for all $i \in J$.*

The third theorem shows how to construct a schedule that meets all due dates if such a schedule exists. The theorem uses the notion of list schedule. A list schedule is a schedule constructed by the following algorithm (known as the list algorithm). Suppose that all tasks are arranged in a list.

1. Among all tasks select a task j with the smallest release time. Let $T = r_j$.
2. Scan the list from left to right and find the first task available for processing in time interval $[T, T+1]$. Assign this task to the first machine and eliminate it from the list.
3. Continue to scan the list until either another task available for processing in the time interval $[T, T+1]$ has been found, or the end of the list has been reached. If another task has been found, assign this task to the second machine and eliminate it from the list. Otherwise leave the second machine idle.
4. If all tasks have been assigned, then halt. Otherwise among all tasks, which have not been assigned for processing, select task i with the smallest completion time. Set $T = \max\{T+1, r_i\}$ and go to Step 2.

Theorem 3 *Let $\{D_i : i \in J\}$ be a set of consistent due dates. Then any list schedule α for the set J, corresponding to a list where tasks are arranged in a nondecreasing order of consistent due dates D_i, meets these due dates, i.e. $C_i(\alpha) \leq D_i$, for all $i \in J$.*

The algorithm described in this section constructs an ideal schedule, denoted δ, in iterative manner. At each iteration the procedure determines a fragment of δ referred to as a block. A task can be included in a block only if all its predecessors have been already included in the same or previously constructed

blocks. Each block is a fragment of some list schedule. In constructing this list schedule the procedure associates with each task i, which has not been included in the previously constructed blocks, a positive integer μ_i, referred to as a priority, and forms a list by arranging tasks in the decreasing order of their priorities (no two tasks have the same priority). Although both criteria C_{max} and C_Σ do not assume any due dates, the calculation of priorities is based on some consistent due dates. The subroutine, which calculates priorities, will be referred to as the μ-algorithm. In order to describe the μ-algorithm it is convenient to introduce the following notation. For an arbitrary integer t and an arbitrary task i, $K(i,t)$ will denote the set of all tasks j such that $r_j < t$ and $i \rightarrow j$. Suppose that $K(i,t) \neq \emptyset$ and each task $j \in K(i,t)$ has been already assigned its priority μ_j. Denote by $\omega(i,t)$ the $|K(i,t)|$-dimensional vector $(\mu_{j_1}, \ldots, \mu_{j_{|K(i,t)|}})$, where $\mu_{j_1} > \cdots > \mu_{j_{|K(i,t)|}}$ and $j_k \in K(i,t)$ for all $1 \leq k \leq |K(i,t)|$. In other words, $\omega(i,t)$ lists all successors j of task i, satisfying the condition $r_j < t$, in the decreasing order of their priorities.

Let $J \subseteq \{1, \ldots, n\}$ be the set of all tasks that have not been included in the previously constructed blocks. In constructing a new block, only these tasks are considered and all tasks which have been already included in the previously constructed blocks are ignored. The construction of the first block and some other blocks will require calculating for each task $i \in J$ a priority μ_i using the following algorithm.

μ-algorithm

1. Set $a = 1$.
2. Among all tasks from J, which have not been assigned their priorities, select a task with the largest due date. Denote this due date by d. If several tasks from J, which have not been assigned their priorities, have due dates equal to d, select among them any task without successors. If every task $i \in J$ with $d_i = d$, which has not been assigned μ_i, has a successor, select among these tasks any task with the smallest in the lexicographical order vector $\omega(i,d)$. Let u be the selected task.
3. Set $\mu_u = a$ and $a = a + 1$. If there is a task in J, which has not been assigned its priority, go to step 2. Otherwise halt.

The idea behind the algorithm, constructing an ideal schedule, can be described as follows. Suppose that we know that for any instance of the considered scheduling problem there exists an ideal schedule. Since any ideal schedule is optimal for both C_{max} and C_Σ, even without knowing any ideal schedule, it is not difficult to find integers d_j such that

$$C_j(\nu) \leq d_j, \qquad \text{for some ideal schedule } \nu \text{ and all } j \in N \tag{5}$$

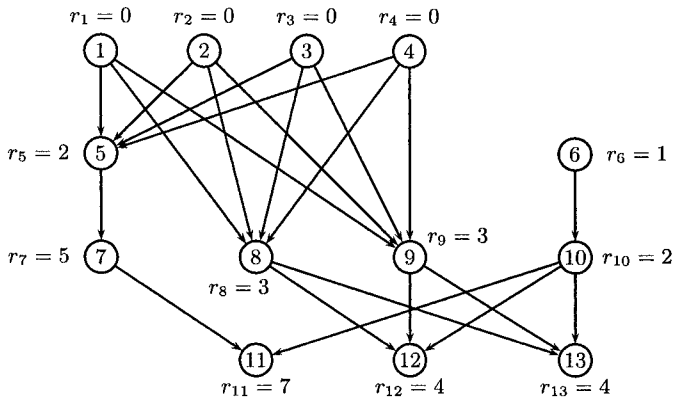


Figure 3. Partially ordered set of tasks.

By (5), the set of schedules that meet the due dates d_j is not empty. Therefore, using Garey and Johnson (1977), it is possible to construct a schedule, say σ, such that $C_j(\sigma) \leq d_j$ for all $j \in N$. If due dates d_j are sufficiently tight, σ coincides with an ideal schedule. The Main Algorithm, described below, is an iterative procedure that at each iteration either tightens the current due dates, or determines that these due dates are already tight enough and specifies a fragment of an ideal schedule.

The following example illustrates this idea. Consider the partially ordered set of tasks depicted in Figure 3, where nodes represent tasks and arrows represent precedence constraints. The corresponding release time is given next to each node.

It can be shown that for the considered problem there exists an ideal schedule (a proof that such a schedule exists for any instance of the problem with two parallel identical machines, precedence constraints, and unit execution time tasks with release times is given in the next section and is closely related to the analysis of the Main Algorithm). Denote this ideal schedule by ν. Observe that the partially ordered set of tasks is comprised of 13 tasks. Taking into account that ν is optimal for both C_{max} and C_Σ, it is easy to see that (5) holds if $d_j = r_j + 13$ for all $1 \leq j \leq 13$. The priorities μ_j, calculated for these d_j according to the μ-algorithm, are presented in the table below.

task j	1	2	3	4	5	6	7	8	9	10	11	12	13
μ_j	13	12	11	10	7	9	2	5	6	8	1	3	4

For example, we have $d_5 = 15$ and $d_{10} = 15$. Since according to the precedence constraints both 5 and 10 have successors, in order to determine μ_5 and

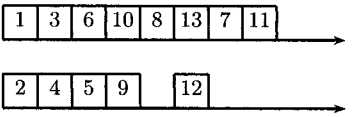


Figure 4.

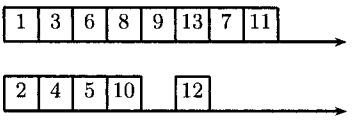


Figure 5.

μ_{10} we need to compare vectors $\omega(5, 15)$ and $\omega(10, 15)$. Because $\omega(5, 15) = (2, 1)$ is smaller in the lexicographical order than $\omega(10, 15) = (4, 3, 1)$, $\mu_5 = 7$ whereas $\mu_{10} = 8$.

Consider the list schedule σ, corresponding to the list 1, 2, 3, 4, 6, 10, 5, 9, 8, 13, 12, 7, 11, where tasks are arranged in the decreasing order of μ_j. Schedule σ is depicted in Figure 4.

The smallest integer $0 < t \leq C_{max}(\sigma)$ such that at most one task is processed in the time interval $[t-1, t]$ is $t = 5$. By the list algorithm, each task j, satisfying the inequality $C_8(\sigma) < C_j(\sigma)$, either is a successor of task 8 in the precedence constraints or has $r_j \geq 5$. Therefore, in any schedule, where task 8 has the completion time greater than $C_8(\sigma)$, only tasks 1, 2, 3, 4, 6, 5, 10 and 9 can be processed in the time interval $[0, C_8(\sigma)]$. Hence, this schedule cannot be optimal for the criterion C_Σ, because in σ task 8 is also processed in this time interval. Therefore $C_8(\nu) \leq C_8(\sigma) = 5$, and the replacement of $d_8 = 16$ by $d_8 = 5$ does not violate the inequality $C_8(\nu) \leq d_8$.

The table below presents consistent due dates D_j, calculated according to Garey and Johnson (1977) for the new set of due dates $\{d_j : 1 \leq j \leq 13\}$ that differs from the original one only by the value of d_8 which now equals 5. By Theorem 2, $C_j(\nu) \leq D_j$ for all $1 \leq j \leq 13$. The above-mentioned table also contains new μ_j, calculated according to the μ-algorithm for the set of consistent due dates $\{D_j : 1 \leq j \leq 13\}$.

task j	1	2	3	4	5	6	7	8	9	10	11	12	13
D_j	4	4	4	4	15	14	18	5	16	15	20	17	17
μ_j	13	12	11	10	6	8	2	9	5	7	1	3	4

Again, denote by σ the list schedule, corresponding to the list where tasks are arranged in the decreasing order of μ_j. This schedule is depicted in Figure 5.

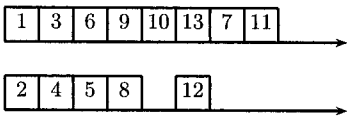


Figure 6.

Using σ and reasoning analogous to that presented above for the task 8, it is easy to see that $C_9(\nu) \leq C_9(\sigma) = 5$. Therefore, the set of due dates $\{d_j : 1 \leq j \leq 13\}$, where $d_j = D_j$ for all $j \neq 9$ and $d_9 = 5$, satisfies (5). The table below presents consistent due dates $\{D_j : 1 \leq j \leq 13\}$, calculated according to Garey and Johnson (1977) for this new set $\{d_j : 1 \leq j \leq 13\}$. By Theorem 2, these new consistent due dates satisfy the inequalities $C_j(\nu) \leq D_j$ for all $1 \leq j \leq 13$. Analogously to the previous iteration, the table below also contains new μ_j, calculated according to the μ-algorithm for the new set of consistent due dates $\{D_j : 1 \leq j \leq 13\}$.

task j	1	2	3	4	5	6	7	8	9	10	11	12	13
D_j	4	4	4	4	15	14	18	5	5	15	20	17	17
μ_j	13	12	11	10	5	7	2	8	9	6	1	3	4

Again, let σ be the list schedule, corresponding to the list where tasks are arranged in the decreasing order of μ_j. This schedule is depicted in Figure 6.

Analogously to the above, σ indicates that $C_{10}(\nu) \leq C_{10}(\sigma) = 5$. Therefore, the set of due dates $\{d_j : 1 \leq j \leq 13\}$, where $d_j = D_j$ for all $j \neq 10$ and $d_{10} = 5$, satisfies (5). The new set $\{d_j : 1 \leq j \leq 13\}$ leads to a new set of consistent due dates $\{D_j : 1 \leq j \leq 13\}$, calculated according to Garey and Johnson (1977). These new due dates D_j are presented in the following table.

task j	1	2	3	4	5	6	7	8	9	10	11	12	13
D_j	4	4	4	4	15	4	18	5	5	5	20	17	17

By Theorem 2, these new consistent due dates satisfy the inequalities $C_j(\nu) \leq D_j$ for all $1 \leq j \leq 13$. In other words, now we know that $C_8(\nu) \leq 5$, $C_9(\nu) \leq 5$, $C_{10}(\nu) \leq 5$, $C_1(\nu) \leq 4$, $C_2(\nu) \leq 4$, $C_3(\nu) \leq 4$, $C_4(\nu) \leq 4$ and $C_6(\nu) \leq 4$. In order to decide whether these consistent due dates are tight enough to determine a fragment of ν, we need to calculate new priorities μ_j and to construct a new list schedule σ. Using $\{D_j : 1 \leq j \leq 13\}$, the μ-algorithm first assigns $\mu_{11} = 1$, and then, in the decreasing order of D_j, $\mu_7 = 2$, $\mu_{12} = 3$, $\mu_{13} = 4$ and $\mu_5 = 5$. Since the next three tasks, 10, 9 and 8, have equal due dates and since $\omega(10,5) = \omega(9,5) = \omega(8,5)$, the priorities 6, 7 and 8 can be assigned to these three tasks in any order. Let $\mu_{10} = 6$, $\mu_9 = 7$, and $\mu_8 = 8$. Now tasks 1, 2, 3, 4 and 6 have equal due dates, but $\omega(1,4) = \omega(2,4) = \omega(3,4) = \omega(4,4) = (8,7,5)$ whereas $\omega(6,4) = (6)$.

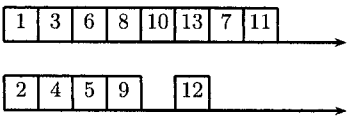


Figure 7.

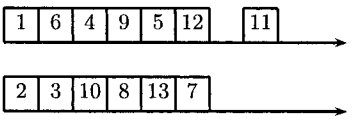


Figure 8.

Because in the lexicographical order (6) is smaller than $(8, 7, 5)$, $\mu_6 = 9$. The priorities 10, 11, 12, 13 can be assigned to the remaining tasks 1, 2, 3 and 4 in any order. Let $\mu_4 = 10$, $\mu_3 = 11$, $\mu_2 = 12$ and $\mu_1 = 13$.

Again, let σ be the list schedule, corresponding to the list where tasks are arranged in the decreasing order of μ_j. This schedule is depicted in Figure 7.

Now $C_{10}(\sigma) = d_{10} = 5$ and we cannot tighten d_{10}. Consider the largest integer $0 < t < C_{10}(\sigma)$, satisfying the condition that there exists a task j such that $C_j(\sigma) = t$ and $\mu_j < \mu_{10}$. It is easy to see that $t = 3$ and $j = 5$. Observe that for $i \in \{8, 9, 10\}$ either $6 \to i$ or $r_i \geq 3$. Hence, in any schedule, where task 6 has the completion time greater than $C_6(\sigma) = 3$, tasks 8–10 are processed in the time interval $[3, \infty]$ and therefore tasks 7, 11, 12 and 13 are processed in the time interval $[5, \infty]$ (see Figures 4–7). Hence, in this schedule only tasks 1–5 can be processed in the time interval $[0, C_6(\sigma)]$. Such a schedule cannot be optimal for the criterion C_Σ, because in σ task 6 is also processed in this time interval. Therefore, $C_6(\nu) \leq C_6(\sigma) = 3$ and the set of due dates $\{d_j : 1 \leq j \leq 13\}$, where $d_j = D_j$ for all $j \neq 6$ and $d_6 = 3$, satisfies (5).

Now the due dates $\{d_j : 1 \leq j \leq 13\}$ are tight enough. Indeed, consider the list schedule, depicted in Figure 8 and corresponding to the list, where tasks are arranged in a nondecreasing order of d_j. It is obvious that the part of this schedule, corresponding to the time interval $[0, 7]$, is a fragment of an ideal schedule (actually, in this example, the entire schedule, obtained in the process of constructing the first fragment, is ideal).

Reasoning in the above example was based on the assumption that an ideal schedule exists. As will be shown in the next section, this assumption is not necessary and can be replaced by two weaker conditions. The remaining part of this section presents the Main Algorithm, which constructs an ideal schedule block by block. The construction of each block commences with determining the starting time of this block, denoted by τ. So, in the above example, we

started with $\tau = 0$. The parameter τ increases from iteration to iteration. So, if τ' and τ'' are two successive values of τ, then the block is a fragment of δ corresponding to the time interval $[\tau', \tau'']$. In the above example, the next value of τ is 7. In what follows, for an arbitrary schedule α and an arbitrary integer t, $R(\alpha, t)$ will denote the set of all tasks i such that $C_i(\alpha) \leq t$.

Main Algorithm

1. Set $J = \{1, \ldots, n\}$, $\tau = \min_{1 \leq i \leq n} r_i$, and $d_i = r_i + n$, for all $1 \leq i \leq n$.

2. Using the procedure from Garey and Johnson (1977) and the set $\{d_i : i \in J\}$, determine the set of consistent due dates $\{D_i : i \in J\}$ such that $D_i \leq d_i$ for each $i \in J$.

3. Using the set of consistent due dates $\{D_i : i \in J\}$ and the μ-algorithm, calculate μ_i for every task $i \in J$. Arrange tasks, constituting the set J, in a list in the decreasing order of their priorities and construct from time point τ a list schedule σ.

4. Select the smallest integer $\tau < t \leq C_{max}(\sigma)$ such that at most one task is processed in the time interval $[t-1, t]$. Denote this integer by $t(\sigma, \tau)$. If $t(\sigma, \tau)$ does not exist or $t(\sigma, \tau) = C_{max}(\sigma)$, then construct the last block of δ by setting $C_i(\delta) = C_i(\sigma)$ for each task $i \in J$ and halt.

5. Among all tasks i satisfying the inequality $C_i(\sigma) > t(\sigma, \tau)$ select a task with the earliest release time and denote this release time by r. If $r \geq t(\sigma, \tau)$, then construct a new block of δ by setting $C_i(\delta) = C_i(\sigma)$ for each task i, satisfying the inequalities $\tau < C_i(\sigma) \leq t(\sigma, \tau)$. This block corresponds to the time interval $[\tau, r]$. Set $\tau = r$, $J = J - R(\delta, r)$ and return to step 4.

6. Set $i(\tau) = j$, where j is the task satisfying the equality $C_j(\sigma) = t(\sigma, \tau)$.

7. If $D_{i(\tau)} > C_{i(\tau)}(\sigma)$, then set $d_{i(\tau)} = C_{i(\tau)}(\sigma)$ and $d_k = D_k$, for all other tasks $k \in J$, and return to step 2. If $D_{i(\tau)} = C_{i(\tau)}(\sigma)$, then select the largest integer $\tau < t < C_{i(\tau)}(\sigma)$ such that at least one task, which is processed in the time interval $[t-1, t]$, has priority less than $\mu_{i(\tau)}$. Denote this integer by t'. If either t' does not exist, or both tasks processed in the time interval $[t'-1, t']$ have priorities less than $\mu_{i(\tau)}$, then construct a new block of δ by setting $C_i(\delta) = C_i(\sigma)$ for each task i, satisfying the inequalities $\tau < C_i(\sigma) \leq t(\sigma, \tau)$. This block corresponds to the time interval $[\tau, t(\sigma, \tau)]$. Set $J = J - R(\delta, t(\sigma, \tau))$. Assign a new release time to every $i \in J$ as the maximum between the old release time and $t(\sigma, \tau)$. Recalculate, if necessary, the release times of all tasks

to satisfy the condition that the relation $i \rightarrow j$ implies the inequality $r_i \leq r_j - 1$. In doing this, consider the release times in increasing order and for any i and j such that $i \rightarrow j$ and $r_i = r_j$ replace r_j by $r_j + 1$. Set $\tau = t(\sigma, \tau)$ and return to step 4.

8. Let task j be the task satisfying the conditions $C_j(\sigma) = t'$ and $\mu_j > \mu_{i(\tau)}$. Set $i(\tau) = j$ and return to step 7.

The validity of the algorithm is established by the following lemma.

Lemma 1 *For any set $\{d_i : i \in J\}$, utilized at step 2, the procedure from Garey and Johnson (1977) constructs a set of consistent due dates $\{D_i : i \in J\}$ satisfying the inequalities $D_i \leq d_i$ for all $i \in J$.*

Proof: By Theorem 1, if for some set of due dates $\{d_i : i \in J\}$ the procedure from Garey and Johnson (1977) fails to determine the set of consistent due dates, then there is no schedule α for the set J satisfying the inequalities $C_i(\alpha) \leq d_i$ for all $i \in J$. Hence, if such a schedule exists, then the procedure from Garey and Johnson (1977) is able to calculate a set of consistent due dates, and it remains to show that for each set of integers $\{d_i : i \in J\}$, utilized at step 2, there exists a schedule α such that $C_i(\alpha) \leq d_i$ for all $i \in J$. It is easy to see that this holds at the first execution of step 2. Suppose that a set of consistent due dates $\{D_i : i \in J\}$ has been determined in accord with step 2 using a set of integers $\{d_i : i \in J\}$. The schedule σ, constructed as a result of the subsequent execution of step 3, is a list schedule corresponding to a list where tasks are arranged in a nondecreasing order of consistent due dates. Then by Theorem 3, $C_i(\sigma) \leq D_i$ for all $i \in J$, and hence at the next execution of step 2, $C_i(\sigma) \leq d'_i$ for all $i \in J'$, where J' is the set of tasks and $\{d'_i : i \in J'\}$ is the set of due dates considered at this execution of step 2. □

2.2 Proof of Optimality

The execution of any step of the main algorithm, apart from the very first, starts with some value of the parameter τ, some set of tasks J, and some set of due dates $\{d_i : i \in J\}$ assigned either during the execution of step 1 or step 7. For the purpose of the following discussion, it is convenient to introduce the notion of a regular call. In conditions (c1) and (c2), stated below, τ is the value of this parameter at the start of execution of the considered step, J is the set of tasks which have not been included in the blocks constructed prior to this call, and $\{d_i : i \in J\}$ are due dates associated with tasks from J at the moment of this call. A call of a step is regular if

(c1) there exists a schedule η for the set N, which is optimal for the criterion C_{max}, coincides with δ on the time interval $[\min_{1 \leq i \leq n} r_i, \tau]$, and satisfies inequalities $C_i(\eta) \leq d_i$ for all $i \in J$;

(c2) there exists a schedule β for the set N, which is optimal for the criterion C_Σ, coincides with δ on the time interval $[\min_{1\le i\le n} r_i, \tau]$, and satisfies inequalities $C_i(\beta) \le d_i$ for all $i \in J$.

Lemma 2 *If the Main Algorithm terminates at a regular call of step 4, then δ is an ideal schedule.*

Proof: According to (c1) and (c2), schedule δ coincides with β and η in the time interval $[\min_{1\le i\le n} r_i, \tau]$, and these two schedules are optimal for the criteria C_Σ and C_{max}, respectively. The tasks, which are not processed in this time interval, form the set J and are processed in the current schedule σ in the most efficient manner from the viewpoint of both criteria, since in this schedule both machines are busy in the time interval $[\tau, C_{max}(\sigma) - 1]$. Therefore, the resulting schedule δ is optimal for both criteria C_Σ and C_{max}. □

The following lemmas show that any regular call results in another regular call.

Lemma 3 *If a call of step 5 is regular, then the next call is also regular.*

Proof: If the next call is a call of step 6, then this new call is also regular, because in this case neither value of τ, nor sets J and $\{d_i : i \in J\}$ are changed.

Suppose that the call of step 5 results in a call of step 4, and let τ, J and $\{d_i : i \in J\}$ be the value of the parameter τ, the set of tasks and the set of due dates corresponding to this call of step 5. Let J' be the set of tasks which remain not included in the already constructed blocks after the completion of step 5. Since the call of step 5 is regular, there exists a schedule β, specified in (c2). Construct a new schedule β' by setting

$$C_i(\beta') = \begin{cases} C_i(\sigma), & \text{for all } \ i \in R(\sigma, t(\sigma,\tau)) \\ C_i(\beta), & \text{for all } \ i \notin R(\sigma, t(\sigma,\tau)) \end{cases}$$

where $t(\sigma,\tau)$ and σ are the point in time and the list schedule considered during the execution of step 5. Since $r_i \ge t(\sigma,\tau)$ for all $i \in J - R(\sigma, t(\sigma,\tau))$,

$$R(\beta, t(\sigma,\tau)) \subseteq R(\delta,\tau) \cup R(\sigma, t(\sigma,\tau)) = R(\beta', t(\sigma,\tau))$$

This relation together with the observation that in β' both machines are occupied in the time interval $[\tau, t(\sigma,\tau) - 1]$ leads to the conclusion that schedule β' is optimal for the criterion C_Σ.

On the other hand, step 5 does not change the set of due dates $\{d_i : i \in J\}$, and the inequalities $C_i(\beta) \le d_i$ and $C_i(\sigma) \le d_i$, which hold for all $i \in J$, imply that $C_i(\beta') \le d_i$ for all $i \in J'$. Hence, (c2) holds after the completion of step 5.

Let η be a schedule specified in (c1). Then the result that (c1) holds after the completion of step 5 can be proven analogously to the one above by constructing a new schedule η', where

$$C_i(\eta') = \begin{cases} C_i(\sigma), & \text{for all } \ i \in R(\sigma, t(\sigma,\tau)) \\ C_i(\eta), & \text{for all } \ i \notin R(\sigma, t(\sigma,\tau)) \end{cases}$$

So, the call of step 4 is a regular one. □

Suppose that a regular call of step 7 is followed by either a call of step 2 or a call of step 4. Suppose that this call of step 7 corresponds to some τ, J, $\{d_i : i \in J\}$, $\{D_i : i \in J\}$, σ, and $i(\tau)$.

Lemma 4 *Let α be an arbitrary schedule for J such that $C_v(\alpha) \leq d_v$, for all $v \in J$, and let α satisfy at least one of the following two conditions: either $C_{i(\tau)}(\alpha) \geq C_{i(\tau)}(\sigma)$, or there exists $x \in J$ such that $\mu_x > \mu_{i(\tau)}$ and $C_x(\alpha) = D_{i(\tau)}$. Then*

$$R(\alpha, t(\sigma,\tau)) \subseteq R(\sigma, t(\sigma,\tau)) \tag{6}$$

Proof: Suppose that the considered call of step 7 is the kth call of this step since the last call of step 6. Consider a sequence of tasks $g_1, \ldots, g_k$ and a sequence of sets of tasks $J^0, J^1, \ldots, J^k$ determined as follows. Task g_1 satisfies the equality $C_{g_1}(\sigma) = t(\sigma,\tau)$. Each task g_i, $1 < i \leq k$, is the task selected as a result of the $(i-1)$st call of step 8. Set J^0 is the set of all tasks i satisfying the inequality $C_i(\sigma) > C_{g_1}(\sigma)$. Each J^i, $1 \leq i < k$, is comprised of task g_i and all tasks v such that $C_{g_{i+1}}(\sigma) < C_v(\sigma) < C_{g_i}(\sigma)$. Set J^k is defined as follows. If $C_{g_k}(\sigma) < D_{g_k}$, then $J^k = \{g_k\}$. If $C_{g_k}(\sigma) = D_{g_k}$, then J^k depends on whether or not t' has been calculated as a result of the kth call of step 7. If t' has been calculated, then J^k is comprised of task g_k and all tasks v such that $t' < C_v(\sigma) < C_{g_k}(\sigma)$. If t' has not been calculated, then J^k is comprised of task g_k and all tasks v such that $\tau < C_v(\sigma) < C_{g_k}(\sigma)$.

Consider a sequence of tasks $y_1, \ldots, y_k$, where for all $1 \leq i \leq k$

$$C_{y_i}(\alpha) = \max_{v \in J^i} C_v(\alpha)$$

For any task y_i, D_{y_i} and the corresponding D_{g_i} satisfy the inequality $D_{y_i} \leq D_{g_i}$, because $\mu_{y_i} \geq \mu_{g_i}$. On the other hand, $C_v(\alpha) \leq d_v$ for all $v \in J$, and therefore by Theorem 2, $C_{y_i}(\alpha) \leq D_{y_i}$. Hence, $C_{y_i}(\alpha) \leq D_{g_i}$ for all $1 \leq i \leq k$.

Observe that $g_1 \rightarrow v$ for all $v \in J^0$ such that $r_v < C_{g_1}(\sigma)$, because σ is a list schedule and one of the machines is idle during the time interval $[C_{g_1}(\sigma) - 1, C_{g_1}(\sigma)]$. Moreover, for all $1 < i \leq k$, $g_i \rightarrow v$ for all $v \in J^{i-1}$ such that $r_v < C_{g_i}(\sigma)$, because the list corresponding to σ arranges tasks in the decreasing

order of their priorities and during the time interval $[C_{g_i}(\sigma) - 1, C_{g_i}(\sigma)]$ one of the machines processes a task with a priority less then priorities of tasks in J^{i-1}. Therefore, if $C_{g_i}(\alpha) \geq C_{g_i}(\sigma)$, then

$$\min_{v \in J^{i-1}} C_v(\alpha) \geq \min_{v \in J^{i-1}} C_v(\sigma) \tag{7}$$

Suppose that there exists a task q such that $\mu_q > \mu_{g_i}$ and $C_q(\alpha) = D_{g_i}$. By Theorem 2, $C_q(\alpha) \leq D_q$. The inequality $\mu_q > \mu_{g_i}$ implies that $D_q \leq D_{g_i}$, which together with $D_{g_i} = C_q(\alpha) \leq D_q$ gives $D_q = D_{g_i}$. Since $D_q = D_{g_i}$ and $\mu_q > \mu_{g_i}$, $\omega(q, D_q)$ is greater than or equal to $\omega(g_i, D_{g_i})$ in the lexicographical order. This is equivalent to the statement that $q \rightarrow v$ for all $v \in J^{i-1}$ such that $r_v < C_{g_i}(\sigma)$, which again gives (7).

The above reasonings lead to the conclusion that (6) holds if $k = 1$, because in this case $g_1 = i(\tau)$ and consequently either $C_{g_1}(\alpha) \geq C_{g_1}(\sigma)$, or $C_x(\alpha) = D_{g_1}$. Suppose that $k > 1$, then $C_{g_i}(\alpha) = D_{g_i}$ for all $1 \leq i < k$ and $g_k = i(\tau)$. Since either $C_{g_k}(\alpha) \geq C_{g_k}(\sigma)$, or there exists $x \in J$ such that $\mu_x > \mu_{g_k}$ and $C_x(\alpha) = D_{g_k}$,

$$\min_{v \in J^{k-1}} C_v(\alpha) \geq \min_{v \in J^{k-1}} C_v(\sigma)$$

Therefore,

$$D_{g_{k-1}} = C_{g_{k-1}}(\sigma) \leq C_{y_{k-1}}(\alpha) \leq D_{g_{k-1}},$$

which gives $C_{y_{k-1}}(\alpha) = D_{g_{k-1}}$. On the other hand, for any $1 < i < k$, either $y_i = g_i$, or $y_i \neq g_i$ and $\mu_{y_i} > \mu_{g_i}$. Therefore the equality $C_{y_i}(\alpha) = D_{g_i}$ implies (7), which analogously to the above gives $C_{y_{i-1}}(\alpha) = D_{g_{i-1}}$. Consequently, $C_{y_{k-1}}(\alpha) = D_{g_{k-1}}$ implies $C_{y_1}(\alpha) = D_{g_1}$, which in turn gives

$$\min_{v \in J^0} C_v(\alpha) \geq \min_{v \in J^0} C_v(\sigma)$$

and therefore (6). □

Lemma 5 *If a call of step 7 is regular, then the next call is also regular.*

Proof: Let a regular call of step 7 correspond to some τ, J, $\{d_i : i \in J\}$, $\{D_i : i \in J\}$, $i(\tau)$ and σ. Since this call is regular, there exist schedules η and β specified in (c1) and (c2). If this call is followed by a call of step 8, then the latter call is also regular, because in this case neither the parameter τ, nor the sets J and $\{d_i : i \in J\}$ change during the execution of step 7.

Suppose that the call of step 7 is followed by a call of step 2. Then step 7 results in the replacement of $\{d_i : i \in J\}$ by due dates $\{d'_i : i \in J\}$, where $d_{i(\tau)} = C_{i(\tau)}(\sigma)$ and $d_v = D_v$ for all other $v \in J$. If $C_{i(\tau)}(\eta) \leq C_{i(\tau)}(\sigma)$ and $C_{i(\tau)}(\beta) \leq C_{i(\tau)}(\sigma)$, then $C_{i(\tau)}(\eta) \leq d'_{i(\tau)}$ and $C_{i(\tau)}(\beta) \leq d'_{i(\tau)}$. On the other hand, by Theorem 2, $C_v(\eta) \leq D_v = d'_v$ and $C_v(\beta) \leq D_v = d'_v$ for all other $v \in J$. Hence, in this case, the call of step 2 is also regular.

If $C_{i(\tau)}(\beta) > C_{i(\tau)}(\sigma)$, then consider schedule β', where

$$C_i(\beta') = \begin{cases} C_i(\sigma), & \text{for all } \ i \in R(\sigma, t(\sigma,\tau)) \\ C_i(\beta), & \text{for all } \ i \notin R(\sigma, t(\sigma,\tau)) \end{cases} \tag{8}$$

By Lemma 4

$$J \cap R(\beta, t(\sigma,\tau)) \subseteq R(\sigma, t(\sigma,\tau)) \subseteq R(\beta', t(\sigma,\tau))$$

Since in σ both machines are busy in the interval $[\tau, t(\sigma,\tau) - 1]$, $C_\Sigma(\beta') \leq C_\Sigma(\beta)$. Therefore, schedule β' is optimal for the criterion C_Σ. On the other hand, $C_{i(\tau)}(\sigma) = d'_{i(\tau)}$, and by Theorem 3, $C_i(\sigma) \leq D_i = d'_i$ for all other $i \in R(\sigma, t(\sigma,\tau))$. Furthermore, by Theorem 2, $C_i(\beta) \leq D_i = d'_i$ for all $i \in J$ such that $i \notin R(\sigma, t(\sigma,\tau))$. Therefore, $C_v(\beta') \leq d'_v$ for all $v \in J$, and β' satisfies (c2) for $\{d'_v : v \in J\}$.

Similarly, if $C_{i(\tau)}(\eta) > C_{i(\tau)}(\sigma)$, then, at the moment of the call of step 2, (c1) holds for schedule η', where

$$C_i(\eta') = \begin{cases} C_i(\sigma), & \text{for all } \ i \in R(\sigma, t(\sigma,\tau)) \\ C_i(\eta), & \text{for all } \ i \notin R(\sigma, t(\sigma,\tau)) \end{cases} \tag{9}$$

Hence, the call of step 2 is regular.

Suppose that the call of step 7 is followed by a call of step 4. Consider schedule β' defined by (8). Let J' be the set of all tasks v satisfying the inequality $C_v(\beta') > t(\sigma,\tau)$. Since β satisfies (c2), $C_v(\beta') \leq d_v$ for all $v \in J'$, and in order to prove that β' satisfies (c2) at the moment of the call of step 4, it remains to show that β' is optimal for the criterion C_Σ. Let

$$T = \begin{cases} t', & \text{if } t' \text{ has been found during the execution of step 7} \\ \tau, & \text{otherwise} \end{cases}$$

Consider the set of tasks $\tilde{J}$ which is comprised of $i(\tau)$ and all tasks v such that $T < C_v(\sigma) < C_{i(\tau)}(\sigma)$. It is easy to see that $C_v(\beta) > T$ for all $v \in \tilde{J}$. On the other hand, for any $v \in \tilde{J}$, $\mu_v \geq \mu_{i(\tau)}$ and therefore $D_v \leq D_{i(\tau)}$. Since, by Theorem 2, $C_v(\beta) \leq D_v$, all tasks constituting $\tilde{J}$ are processed in β in the time interval $[T, D_{i(\tau)}]$. Hence $C_v(\beta) = D_{i(\tau)}$ for some $v \in \tilde{J}$. Then again by Lemma 4

$$J \cap R(\beta, t(\sigma,\tau)) \subseteq R(\sigma, t(\sigma,\tau)) \subseteq R(\beta', t(\sigma,\tau))$$

and because both machines are busy in the time interval $[\tau, t(\sigma,\tau) - 1]$, so $C_\Sigma(\beta') \leq C_\Sigma(\beta)$, which implies that β' is optimal for the criterion C_Σ. Thus, at the moment of the call of step 4, β' satisfies (c2). The proof that η', defined by (9), satisfies (c1) is similar. Hence, the call of step 4 is regular. □

Theorem 4 *The procedure constructs an ideal schedule and has complexity* $O(n^6)$.

Proof: Each block of schedule δ corresponds to a particular value of the parameter τ and this parameter takes on at most n different values. Each d_i satisfies inequalities $r_i < d_i \leq r_i + n$ and if its value changes, then a new value is at least one unit less than the previous one. Therefore, the number of calls of step 2 is $O(n^2)$. Since the calculation of consistent due dates according to step 2 requires $O(n^3)$ operations, the total complexity is $O(n^6)$.

It is easy to see that the first call of steps 2 is regular. On the other hand, Lemmas 3 and 5 guarantee that all subsequent calls are regular too. Therefore, by Lemma 2, the resulting schedule is ideal. □

3. COMPUTATIONAL COMPLEXITY OF THE $P2|in\text{-}tree, p_j = 1, size_j|C_\Sigma$ PROBLEM

This section shows that the problem $P2|in\text{-}tree, p_j = 1, size_j|C_\Sigma$ is NP-hard in the strong sense by a reduction of the 3-partition problem which is NP-complete in the strong sense (Garey and Johnson, 1979). The 3-partition problem can be stated as follows:

> *Instance:* A set of positive integers $\mathcal{A} = \{a_1, \ldots, a_{3z}\}$ together with a positive integer b such that $\sum_{i=1}^{3z} a_i = zb$ and $\frac{b}{4} < a_i < \frac{b}{2}$, for all $i \in \{1, \ldots, 3z\}$.
>
> *Question:* Does there exist a partition of the set $\mathcal{A}$ into subsets $\mathcal{A}_1, \ldots, \mathcal{A}_z$, such that $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for all $i \neq j$, and the sum of the elements of each subset is equal to b?

For any instance of the 3-partition problem, the corresponding scheduling decision problem will contain $3z$ chains (one for each integer in $\mathcal{A}$), an in-tree, and a set of 2-tasks. Following Brucker *et al.* (2000), a task j requiring $size_j$ machines is referred to as a $size_j$-task. Analogously to Zinder *et al.* (submitted), for each integer $a_j \in \mathcal{A}$, the corresponding scheduling problem contains a chain of $2a_j$ tasks, where the first a_j tasks in the chain are 2-tasks, and the remaining tasks are 1-tasks. The chain for some $a_i \in \mathcal{A}$ and the in-tree are depicted in Figure 9. The set of a_j 2-tasks associated with a_j will be denoted by M_1^j. Similarly, the set of all remaining tasks associated with a_j will be denoted by M_2^j. Let $M_1 = \cup_{j=1}^{3z} M_1^j$ and $M_2 = \cup_{j=1}^{3z} M_2^j$. Thus, all tasks in M_1 are 2-tasks, whereas all tasks in M_2 are 1-tasks.

In contrast to Zinder *et al.* (submitted), where the out-tree is comprised of 1-tasks only, the in-tree contains both 1-tasks and 2-tasks. It is convenient to group these tasks into several sets denoted K_v^i, where K_1^i for $1 \leq i \leq z$, K_2^i for $1 \leq i \leq z-1$, and K_3^i for $1 \leq i \leq z-1$ consist of 1-tasks, and K_4^i for

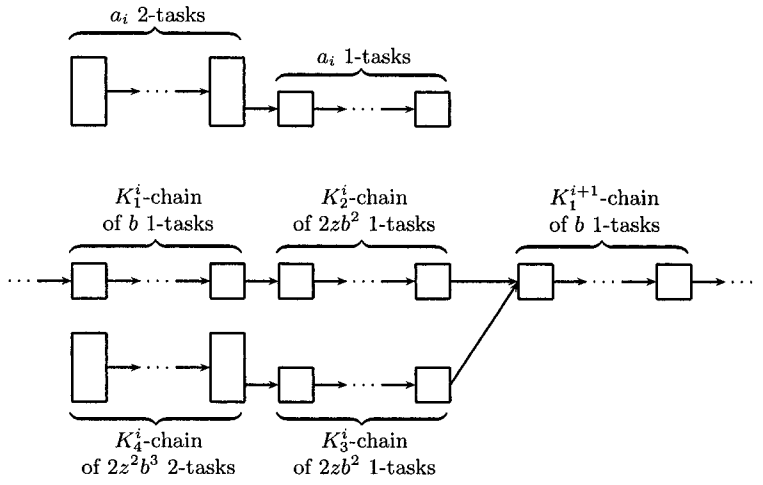


Figure 9.

$1 \leq i \leq z-1$ consist of 2-tasks. These sets have the following cardinalities: $|K_1^i| = b$, $|K_2^i| = |K_3^i| = 2zb^2$, $|K_4^i| = 2z^2b^3$. The tasks constituting each set form a chain. These chains will be referred to as K_v^i-chains and are linked as follows. For each $1 \leq i \leq z-1$

- the last task in the K_1^i-chain precedes the first task in the K_2^i-chain;
- the last task in the K_4^i-chain precedes the first task in the K_3^i-chain;
- the first task in the K_1^{i+1}-chain has two immediate predecessors: the last task in the K_2^i-chain and the last task in the K_3^i-chain.

Let $K_1 = \cup_{i=1}^{z} K_1^i$, $K_2 = \cup_{i=1}^{z-1} K_2^i$, $K_3 = \cup_{i=1}^{z-1} K_3^i$, and $K_4 = \cup_{i=1}^{z-1} K_4^i$, and let K_5 be the set of 2-tasks, where

$$|K_5| = 2 \sum_{j=1}^{|M_1|+|K_1|+|K_2|+|K_4|} j$$

and each task of the set K_5 does not precede nor succeed any other task. Then the set of all tasks is $N = M_1 \cup M_2 \cup K_1 \cup K_2 \cup K_3 \cup K_4 \cup K_5$.

Let $\widehat{\sigma}$ be a schedule, probably infeasible, satisfying the following conditions:

(t1) for each $1 \leq i \leq z$, b tasks from M_1 are processed in the time interval $[(2b + 2zb^2 + 2z^2b^3)(i-1), (2b + 2zb^2 + 2z^2b^3)(i-1) + b]$;

(t2) for each $1 \leq i \leq z$, b tasks from K_1^i and b tasks from M_2 are processed in parallel in the time interval $[(2b + 2zb^2 + 2z^2b^3)(i-1) + b, (2b + 2zb^2 + 2z^2b^3)(i-1) + 2b]$;

(t3) for each $1 \leq i \leq z-1$, $2z^2b^3$ tasks from K_4^i are processed in the time interval $[(2b+2zb^2+2z^2b^3)(i-1)+2b, (2b+2zb^2+2z^2b^3)(i-1)+2b+2z^2b^3]$.

(t4) for each $1 \leq i \leq z-1$, $2zb^2$ tasks from K_2^i and $2zb^2$ tasks from K_3^i are processed in parallel in the time interval $[(2b + 2zb^2 + 2z^2b^3)(i-1) + 2b + 2z^2b^3, (2b + 2zb^2 + 2z^2b^3)i]$.

(t5) all tasks from K_5 are processed in the time interval $[(2b + 2zb^2 + 2z^2b^3)(z-1) + 2b, (2b + 2zb^2 + 2z^2b^3)(z-1) + 2b + |K_5|]$.

Let $C = \sum_{j \in N} C_j(\widehat{\sigma})$. Then the scheduling decision problem is a problem requiring to answer the question: does there exist a feasible schedule σ satisfying the inequality $C_\Sigma(\sigma) \leq C$?

In what follows, for any positive integer t the time interval $[t-1, t]$ will be called a time slot t.

Lemma 6 *For any schedule σ, optimal for the criterion C_Σ,*

$$\max_{j \in N} C_j(\sigma) = |M_1| + |K_1| + |K_2| + |K_4| + |K_5| \tag{10}$$

and

$$\max_{j \in N - K_5} C_j(\sigma) + 1 = \min_{j \in K_5} C_j(\sigma) \tag{11}$$

Proof: Let σ be an arbitrary optimal schedule. It follows from the optimality of σ that the time interval $[0, \max_{j \in N} C_j(\sigma)]$ does not contain any time slot in which both machines are idle. Suppose that σ does not satisfy (10). Since each task in K_5 is a 2-task and does not have any successor, σ can be transformed into a schedule η in which all tasks from $N - K_5$ are processed in the same order as in σ, all tasks from K_5 are processed after all tasks from $N - K_5$, and

$$\max_{j \in N} C_j(\eta) = \max_{j \in N} C_j(\sigma)$$

Because the set K_5 is comprised of 2-tasks, the optimality of σ implies the optimality of η, and since σ does not satisfy (10)

$$\max_{v \in N - K_5} C_v(\eta) \geq |M_1| + |K_1| + |K_2| + |K_4| + 1$$

It is easy to see that there exists a feasible schedule μ such that all 2-tasks comprising M_1 are processed in the time interval $[0, |M_1|]$; all 2-tasks constituting K_4 are processed in the time interval $[|M_1|, |M_1| + |K_4|]$; all 1-tasks

from $K_1 \cup K_2$ are processed in the time interval $[|M_1| + |K_4|, |M_1| + |K_4| + |K_1| + |K_2|]$ each in parallel with some task from $M_2 \cup K_3$; and all 2-tasks from K_5 are processed in the time interval $[|M_1| + |K_4| + |K_1| + |K_2|, |M_1| + |K_4| + |K_1| + |K_2| + |K_5|]$.

Because sets M_1 and K_4 are comprised of 2-tasks,

$$\sum_{v \in N - K_5} C_v(\mu) < \sum_{i=1}^{|M_1|+|K_1|+|K_2|+|K_4|} 2i = |K_5|$$

Since

$$\sum_{v \in K_5} C_v(\eta) = |K_5| \max_{v \in N - K_5} C_v(\eta) + 1 + 2 + \cdots + |K_5|$$

and

$$\sum_{v \in K_5} C_v(\mu) = |K_5| \max_{v \in N - K_5} C_v(\mu) + 1 + 2 + \cdots + |K_5|$$

we have

$$\begin{aligned} & C_\Sigma(\eta) - C_\Sigma(\mu) \\ & \quad = \sum_{v \in N - K_5} C_v(\eta) - \sum_{v \in N - K_5} C_v(\mu) + \sum_{v \in K_5} C_v(\eta) - \sum_{v \in K_5} C_v(\mu) \\ & \quad > -|K_5| + |K_5| \left(\max_{v \in N - K_5} C_v(\eta) - \max_{v \in N - K_5} C_v(\mu) \right) \geq 0 \end{aligned}$$

which contradicts the optimality of η, and therefore the optimality of σ.

Let σ be an optimal schedule, then (10) implies that every task from $K_1 \cup K_2$ is processed in σ in parallel with a task from $K_3 \cup M_2$. Hence, the time slot $\max_{j \in N - K_5} C_j(\sigma)$ contains two 1-tasks: the last task in the K_1^z-chain and a task from M_2. Since the set K_5 is comprised of 2-tasks, the optimality of σ implies that all these tasks are processed in σ after the time slot $\max_{j \in N - K_5} C_j(\sigma)$, which is equivalent to (11). □

In what follows, only schedules σ, satisfying (10) and (11), will be considered. For any two tasks j and g, let

$$\delta_{jg}(\sigma) = \begin{cases} 1 & \text{if} \quad C_j(\sigma) < C_g(\sigma) \text{ and } j \text{ is a 2-task} \\ \frac{1}{2} & \text{if} \quad C_j(\sigma) < C_g(\sigma) \text{ and } j \text{ is a 1-task} \\ 1 & \text{if} \quad j = g \\ 0 & \text{otherwise} \end{cases}$$

Let $\Delta_{22}(\sigma)$ be the sum of all $\delta_{jg}(\sigma)$, where both j and g are 2-tasks; $\Delta_{11}(\sigma)$ be the sum of all $\delta_{jg}(\sigma)$, where both j and g are 1-tasks; $\Delta_{12}(\sigma)$ be the sum of

all $\delta_{jg}(\sigma)$, where j is a 1-task and g is a 2-task; and $\Delta_{21}(\sigma)$ be the sum of all $\delta_{jg}(\sigma)$, where j is a 2-task and g is a 1-task.

Since all 1-tasks are processed in σ in pairs, for any $g \in N$,

$$C_g(\sigma) = \sum_{j \in N} \delta_{jg}(\sigma)$$

and consequently,

$$C_\Sigma(\sigma) = \sum_{g \in N} C_g(\sigma) = \sum_{g \in N} \sum_{j \in N} \delta_{jg}(\sigma) = \Delta_{22}(\sigma) + \Delta_{11}(\sigma) + \Delta_{12}(\sigma) + \Delta_{21}(\sigma)$$

It is easy to see that $\Delta_{22}(\sigma) + \Delta_{11}(\sigma)$ is a constant and does not depend on σ. Therefore, for any two schedules α and η, satisfying (10) and (11),

$$C_\Sigma(\alpha) - C_\Sigma(\eta) = \Delta_{12}(\alpha) - \Delta_{12}(\eta) + \Delta_{21}(\alpha) - \Delta_{21}(\eta)$$

Let j be a 1-task and g be a 2-task. Then

$$\delta_{jg}(\alpha) - \delta_{jg}(\eta) = -\frac{1}{2}\{\delta_{gj}(\alpha) - \delta_{gj}(\eta)\}$$

and therefore

$$C_\Sigma(\alpha) - C_\Sigma(\eta) = \frac{1}{2}\{\Delta_{21}(\alpha) - \Delta_{21}(\eta)\} \tag{12}$$

Let σ be an arbitrary schedule. Consider a new schedule η such that

- $C_j(\eta) = C_j(\sigma)$ for all $j \notin K_4 \cup K_3$;
- $\{C_j(\eta) : j \in K_4\} = \{C_j(\sigma) : j \in K_4\}$;
- $C_{j_1}(\eta) < C_{j_2}(\eta)$ for any two tasks $j_1 \in K_4^{i_1}$ and $j_2 \in K_4^{i_2}$ such that $i_1 < i_2$;
- $C_{j_1}(\eta) < C_{j_2}(\eta)$ for any two tasks j_1 and j_2 from the same set K_4^i such that $C_{j_1}(\sigma) < C_{j_2}(\sigma)$;
- $\{C_j(\eta) : j \in K_3\} = \{C_j(\sigma) : j \in K_3\}$;
- $C_{j_1}(\eta) < C_{j_2}(\eta)$ for any two tasks $j_1 \in K_3^{i_1}$ and $j_2 \in K_3^{i_2}$ such that $i_1 < i_2$;
- $C_{j_1}(\eta) < C_{j_2}(\eta)$ for any two tasks j_1 and j_2 from the same set K_3^i such that $C_{j_1}(\sigma) < C_{j_2}(\sigma)$.

It is easy to see that η is feasible and $C_\Sigma(\eta) = C_\Sigma(\sigma)$. In what follows, only schedules η satisfying the conditions

$$\max_{j \in K_4^i} C_j(\eta) < \min_{j \in K_4^{i+1}} C_j(\eta), \quad \text{for all } 1 \le i < z - 1 \tag{13}$$

and

$$\max_{j \in K_3^i} C_j(\eta) < \min_{j \in K_3^{i+1}} C_j(\eta), \quad \text{for all } 1 \leq i < z-1 \tag{14}$$

will be considered.

Let σ be an arbitrary schedule, and suppose that for some i

$$\max_{j \in K_4^i} C_j(\sigma) + 1 < \min_{j \in K_3^i} C_j(\sigma)$$

Let

$$C_{j_1}(\sigma) = \max_{j \in K_4^i} C_j(\sigma) \quad \text{and} \quad C_{j_2}(\sigma) = \min_{j \in K_3^i} C_j(\sigma)$$

Then consider a new schedule η, where

$$C_x(\eta) = \begin{cases} C_x(\sigma) & \text{if } C_x(\sigma) < C_{j_1}(\sigma) \text{ or } C_x(\sigma) \geq C_{j_2}(\sigma) \\ C_x(\sigma) - 1 & \text{if } C_{j_1}(\sigma) < C_x(\sigma) < C_{j_2}(\sigma) \\ C_{j_2}(\sigma) - 1 & \text{if } x = j_1 \end{cases}$$

Since j_1 is a 2-task which has only one immediate successor, task j_2, this schedule is feasible and $C_\Sigma(\eta) \leq C_\Sigma(\sigma)$. In what follows, only schedules η satisfying the condition

$$\max_{j \in K_4^i} C_j(\eta) + 1 = \min_{j \in K_3^i} C_j(\eta), \quad \text{for all } 1 \leq i < z-1 \tag{15}$$

will be considered.

Furthermore, let σ be an arbitrary schedule, and suppose that for some i there exist two tasks $j_1 \in K_4^i$ and $j_2 \in K_4^i$ such that $C_{j_1}(\sigma) + 1 < C_{j_2}(\sigma)$ and $j \notin K_4^i$ for any j satisfying the inequalities $C_{j_1}(\sigma) < C_j(\sigma) < C_{j_2}(\sigma)$. Then consider a new schedule η where

$$C_x(\eta) = \begin{cases} C_x(\sigma) & \text{if } C_x(\sigma) < C_{j_1}(\sigma) \text{ or } C_x(\sigma) \geq C_{j_2}(\sigma) \\ C_x(\sigma) - 1 & \text{if } C_{j_1}(\sigma) < C_x(\sigma) < C_{j_2}(\sigma) \\ C_{j_2}(\sigma) - 1 & \text{if } x = j_1 \end{cases}$$

Since j_1 is a 2-task and j_2 is its only immediate successor, it is obvious that η is a feasible schedule and $C_\Sigma(\eta) \leq C_\Sigma(\sigma)$. Similar reasoning can be used if tasks j_1 and j_2 belong instead of K_4^i to some M_1^i. Therefore, in what follows, only schedules η satisfying the conditions

$$\max_{j \in K_4^i} C_j(\sigma) - \min_{j \in K_4^i} C_j(\sigma) = |K_4^i| - 1, \quad \text{for all } 1 \leq i \leq z-1 \tag{16}$$

and

$$\max_{j \in M_1^i} C_j(\sigma) - \min_{j \in M_1^i} C_j(\sigma) = |M_1^i| - 1, \quad \text{for all } 1 \leq i \leq 3z \tag{17}$$

will be considered.

Lemma 7 *For any schedule η, optimal for the criterion C_Σ, each task from K_1 is processed in parallel with a task from M_2.*

Proof: Suppose that there exists an optimal schedule η which does not satisfy this lemma. Since η is an optimal schedule, by Lemma 6, each task from $K_1 \cup K_2$ is processed in this schedule in parallel with some task from the set $M_2 \cup K_3$. Consider the first time slot containing a task from K_1, say task $u \in K_1^h$, together with a task from K_3, say task v. Suppose that $h > 1$. Since $j \rightarrow g$ for each $j \in \cup_{1 \leq i < h} K_3^i$ and each $g \in K_1^h$, the processing of all tasks constituting $\cup_{1 \leq i < h} K_3^i$ is completed in η before the processing of tasks from K_1^h begins. Each $j \in \cup_{1 \leq i < h} K_3^i$ is processed in η in parallel with some task from $\cup_{1 \leq i < h} K_2^i$, because all tasks constituting $\cup_{1 \leq i < h} K_1^i$ are processed in parallel with tasks from M_2. Since $|\cup_{1 \leq i < h} K_2^i| = |\cup_{1 \leq i < h} K_3^i|$, no other tasks from K_3 are processed in parallel with tasks from $\cup_{1 \leq i < h} K_2^i$. Hence v is the first task in the K_3^h-chain. Suppose that $h = 1$, then any task from K_3 preceding v is to be processed in parallel with some task from K_1^1, which contradicts the selection of v. Hence v is the first task of the K_3^1-chain.

Among all tasks $j \in M_2$, satisfying the condition $C_j(\eta) > C_u(\eta)$, select the task with the smallest completion time. Let it be task w and suppose that this task corresponds to a_q. Let $\widetilde{M}_1^q$ be the set of all tasks $j \in M_1^q$ such that

$$C_u(\eta) < C_j(\eta) < C_w(\eta)$$

By Lemma 6 and by the selection of w, each time slot in the time interval $[C_u(\eta), C_w(\eta) - 1]$, containing a 1-task, contains one task from $K_1 \cup K_2$ and one task from K_3. Let $\widetilde{K}_1$ be the set of all tasks $j \in K_1$ which are processed in η in the time interval $[C_u(\eta), C_w(\eta)]$, $\widetilde{K}_2$ be the set of all tasks $j \in K_2$ which are processed in η in the time interval $[C_u(\eta), C_w(\eta)]$, and $\widetilde{K}_3$ be the set of all tasks $j \in K_3$ which are processed in η in the time interval $[C_u(\eta), C_w(\eta) - 1]$. Let $\widetilde{N}$ be the set of all tasks j such that

$$C_u(\eta) < C_j(\eta) < C_w(\eta) \quad \text{and} \quad j \notin \widetilde{M}_1^q \cup \widetilde{K}_1 \cup \widetilde{K}_2 \cup \widetilde{K}_3$$

Observe that all tasks in $\widetilde{N}$ are 2-tasks.

Among all $j \in \widetilde{K}_3$ select a task with the largest completion time $C_j(\eta)$. Let it be task y and let $y \in K_3^r$ for some $r \geq h$. Then by (14) and the fact that, for any i, $|K_2^i| = |K_3^i|$,

$$C_w(\eta) < \min_{j \in K_1^{r+1}} C_j(\eta) \tag{18}$$

and for any $h \leq i < r$

$$\max_{j \in K_3^i} C_j(\eta) + 1 < \min_{j \in K_1^{i+1}} C_j(\eta) \tag{19}$$

Consider a new schedule α where

- $C_x(\alpha) = C_x(\eta)$ if either $C_x(\eta) < \min\limits_{j \in K_4^h} C_j(\eta)$, or $C_x(\eta) > C_w(\eta)$;
- tasks comprising $\widetilde{M}_1^q$ are processed in the time interval

$$\left[\min_{j \in K_4^h} C_j(\eta) - 1, \min_{j \in K_4^h} C_j(\eta) - 1 + |\widetilde{M}_1^q|\right]$$

  in the same order as in η;
- $C_u(\alpha) = C_w(\alpha) = \min\limits_{j \in K_4^h} C_j(\eta) + |\widetilde{M}_1^q|$;
- tasks comprising K_4^h are processed in the time interval

$$\left[\min_{j \in K_4^h} C_j(\eta) + |\widetilde{M}_1^q|, \min_{j \in K_4^h} C_j(\eta) + |\widetilde{M}_1^q| + |K_4^h|\right]$$

  in the same order as in η;
- tasks comprising $\widetilde{K}_1 \cup \widetilde{K}_2 \cup \widetilde{N}$ are processed in the time interval

$$\left[\min_{j \in K_4^h} C_j(\eta) + |\widetilde{M}_1^q| + |K_4^h|, C_w(\eta)\right]$$

  in the same order as in η;
- tasks comprising $\widetilde{K}_3$ are processed in parallel with tasks from $\widetilde{K}_1 \cup \widetilde{K}_2$ in the same order as in η.

The feasibility of α follows from (16), the selection of w, (18) and (19). On the other hand, by (12),

$$\begin{aligned} C_\Sigma(\alpha) - C_\Sigma(\eta) &= \frac{1}{2}\{\Delta_{21}(\alpha) - \Delta_{21}(\eta)\} \\ &< \frac{1}{2}\left\{-2 \cdot |K_4^h| + (2|\widetilde{K}_3| + 2) \cdot |\widetilde{M}_1^q|\right\} \\ &< \frac{1}{2}\left\{-4z^2b^3 + (4z^2b^2 + 2)\frac{b}{2}\right\} < 0 \end{aligned}$$

which contradicts the optimality of η. □

Theorem 5 *For any instance of the 3-partition problem, a 3-partition exists if and only if there is a feasible schedule η for the corresponding scheduling problem such that $C_\Sigma(\eta) \leq C$.*

Suppose that a 3-partition $\mathcal{A}_1, \ldots, \mathcal{A}_z$ exists, and that for each i, $\mathcal{A}_i = \{a_{i_1}, a_{i_2}, a_{i_3}\}$. Consider a schedule η in which

- for each $1 \leq i \leq z$, tasks from $M_1^{i_1} \cup M_1^{i_2} \cup M_1^{i_3}$ are processed in the time interval $[(2b+2zb^2+2z^2b^3)(i-1), (2b+2zb^2+2z^2b^3)(i-1)+b]$, and tasks from $M_2^{i_1} \cup M_2^{i_2} \cup M_2^{i_3}$ are processed in the time interval $[(2b + 2zb^2 + 2z^2b^3)(i-1) + b, (2b + 2zb^2 + 2z^2b^3)(i-1) + 2b]$ in parallel with tasks from K_1^i;
- tasks from K_5 and from all K_4^i, K_2^i, K_3^i are processed in accord with (t3), (t4) and (t5).

This schedule is feasible, and since η satisfies all the conditions (t1)–(t5), $C_\Sigma(\eta) \leq C$.

Suppose that a 3-partition does not exist. Consider any schedule σ, probably infeasible, satisfying conditions (t1)–(t5), and an optimal schedule η. Denote by $M_i(\sigma)$ the set of all tasks $j \in M_2$ which are processed in σ in parallel with one of the tasks from K_1^i, and denote by $M_i(\eta)$ the set of all tasks $j \in M_2$ which are processed in η in parallel with one of the tasks from K_1^i. It is easy to see that for any $1 \leq i \leq z$ and any $g \in K_1^i \cup M_i(\sigma)$

$$\sum_{j \in M_1 \cup K_4} \delta_{jg}(\sigma) = bi + (i-1)2z^2b^3 \tag{20}$$

and for any $1 \leq i \leq z-1$ and any $g \in K_2^i \cup K_3^i$

$$\sum_{j \in M_1 \cup K_4} \delta_{jg}(\sigma) = bi + 2z^2b^3i \tag{21}$$

Similarly, taking into account (13), (14), (15), (17) and Lemmas 6 and 7, for any $1 \leq i \leq z$ and any $g \in K_1^i \cup M_i(\eta)$

$$\sum_{j \in M_1 \cup K_4} \delta_{jg}(\eta) > (i-1)b + (i-1)2z^2b^3 \tag{22}$$

for any $1 \leq i \leq z-1$ and any $g \in K_2^i \cup K_3^i$

$$\sum_{j \in M_1 \cup K_4} \delta_{jg}(\eta) \geq bi + 2z^2b^3i \tag{23}$$

and for at least one i, say i^*,

$$\sum_{j \in M_1 \cup K_4} \delta_{jg}(\eta) \geq bi^* + 2z^2b^3i^* + 1 \tag{24}$$

Hence by (20)–(24) and (12),

$$C_\Sigma(\eta) - C = C_\Sigma(\eta) - C_\Sigma(\sigma) = \frac{1}{2}\{\Delta_{21}(\eta) - \Delta_{21}(\sigma)\}$$

$$> \frac{1}{2}\{-b|K_1 \cup M_2| + |K_2^{i^*} \cup K_3^{i^*}|\} = \frac{1}{2}\{-2zb^2 + 4zb^2\} > 0$$

Since η is an optimal schedule, there is no feasible schedule with the criterion value less than or equal to C. □

Since the 3-partition problem is NP-complete in the strong sense, Theorem 5 implies that $P2|in\text{-}tree, p_j = 1, size_j|C_\Sigma$ is NP-hard in the strong sense.

4. CONCLUSIONS

The classical Coffman–Graham–Geray result establishing the existence of an ideal schedule for the problem with two identical parallel machines, arbitrary precedence constraints and unit execution time tasks can be extended in two different ways:

- as has been shown above, an ideal schedule exists and can be found in polynomial time if each task has a release time;
- as has been proven in Coffman *et al.* (2003), an ideal schedule exists and can be found in polynomial time if preemptions are allowed.

The question of the existence of an ideal schedule and a polynomial-time algorithm for the model, obtained from the original one by introducing both release times and preemptions, still remains open and can be viewed as a direction of further research.

As has been proven above, the $P2|prec, p_j = 1, size_j|C_\Sigma$ problem is NP-hard in the strong sense even if the precedence constraints are restricted to in-trees. Although this proof together with Zinder *et al.* (submitted), where NP-hardness has been proven for out-trees, strengthens the original result of Brucker *et al.* (2000), these two results do not provide the full characterization of the $P2|prec, p_j = 1, size_j|C_\Sigma$ problem and its complexity status also requires further research.

References

Brucker, P., Knust, S., Roper, D. and Zinder, Y. (2000) Scheduling UET task systems with concurrency on two parallel identical machines. *Mathematical Methods of Operations Research*, **52/3**:369–387.

Coffman Jr, E. G. and Graham, R. L. (1972) Optimal scheduling for two-processor systems. *Acta Informatica*, **1**:200–213.

Coffman, E. G., Sethuraman, J. and Timkovsky, V. G. (2003) Ideal preemptive schedules on two processors. *Acta Informatica*, **39**:597–612.

Garey, M. R. and Johnson, D. S. (1977) Two-processor scheduling with start-time and deadlines. *SIAM Journal of Computing*, **6**:416–426.

Garey, M. R. and Johnson, D. S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B. (1993) Sequencing and scheduling: algorithms and complexity. In *Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin (Eds.), Elsevier, Amsterdam.

Zinder, Y. (1986) An efficient algorithm for deterministic scheduling problem with parallel machines. *Cybernetics*, **N2** (in Russian).

Zinder, Y., Do, V. H. and Oguz, C. (2002) Computational complexity of some scheduling problems with multiprocessor tasks. *Discrete Applied Mathematics*. Submitted.

Personnel Scheduling

TASK SCHEDULING UNDER GANG CONSTRAINTS

Dirk Christian Mattfeld
Technical University of Braunschweig,
Institute of Business Administration,
Spielmannstr. 8,
38106 Braunschweig,
Germany
d.mattfeld@tu-braunschweig.de

Jürgen Branke
University of Karlsruhe,
Institute AIFB,
76128 Karlsruhe,
Germany
branke@aifb.uni-karlsruhe.de

Abstract In this paper, a short-term manpower planning problem is considered where workers are grouped into gangs to support reliable and efficient operations. The goal is to minimise the total number of workers required by determining an appropriate gang structure, assignment of tasks to gangs, and schedule for each gang. We model such a problem as a multi-mode task scheduling problem with time windows and precedence constraints. While the gang structure and assignment of tasks is optimised by a tabu search heuristic, each gang's schedule is generated by solving the corresponding one-machine scheduling problem by an iterated Schrage heuristic. Because the evaluation of a tabu search move is computationally expensive, we propose a number of ways to estimate a move's impact on the solution quality.

Keywords: manpower planning, multi-mode scheduling, gang constraints, precedence constraints, tabu search, local search, one-machine scheduling.

1. INTRODUCTION

We consider a task scheduling problem as it arises: e.g., from the transhipment of finished vehicles (Mattfeld and Kopfer, 2003). The inter-modal split in the logistics chain requires the intermediate storage of vehicles at a stor-

age area of an automobile terminal. The storage or retrieval of a charge of vehicles forms a task. Since a charge can only be retrieved after it has been stored, precedence constraints between pairs of storage and retrieval tasks are introduced. Temporal constraints with respect to the availability of transport facilities are modelled by time windows.

A task can be processed in different modes determined by the number of drivers executing the task. In order to warrant a safe and reliable relocation of vehicles, drivers are grouped into gangs. The gangs do not change over the course of the planning horizon, typically a time span covering a work shift. Therefore all tasks assigned to one gang are processed in the same mode. The number of gangs as well as their sizes are subject to optimisation, as is the sequence of operations within a task. Time windows of tasks and precedence constraints complicate the seamless processing of tasks within a gang. The objective is to minimise the sum of workers over all gangs established.

As long as the same gang processes two tasks coupled by a precedence constraint, gang scheduling can handle the constraint locally. Whenever a precedence relation exists across gang boundaries, it becomes globally visible, and is modelled as a dynamically changing time-window constraint for its associated tasks. If many precedence constraints exist, the seamless utilisation of manpower capacity within the various gangs is massively hindered by dynamically changing time windows. Managing this constraint will be the greatest challenge while searching for a near-optimal solution to the problem.

In this paper we propose a tabu search procedure which moves single tasks between two gangs. The performance of a move requires the re-scheduling of the two gangs involved. The associated sub-problems are modelled as one-machine problems with heads and tails and varying modes of processing. Since such a sub-problem is already NP-hard for a single mode of processing (Carlier, 1982), an efficient base-heuristic is iteratively applied to determine the manpower demand. Because the evaluation of a move's impact on the solution quality is computationally expensive, the selection of a move in the tabu search heuristic is based on estimates of the move's impact on the manpower.

In Section 2 we discuss related problems and develop a mathematical model. In Section 3 we describe the algorithm in detail, namely, the tabu search framework, the neighbourhood definition, the procedure of scheduling a gang, and, finally, the approximation proposed for selecting a move. We perform a computational investigation for a set of problem parameters in Section 4 before we conclude.

2. RELATED WORK AND PROBLEM MODELLING

We first describe some related problems before we develop a model for the problem considered.

The consideration of multiple modes in the resource constrained project scheduling allows a trade-off between a task's processing time and its resource consumption (Brucker *et al.*, 1999). Mode-identity constraints for prescribed subsets of tasks have been introduced in order to allow the assignment of identical personnel to a group of related tasks (Salewski *et al.*, 1997).

In the audit staff scheduling problem, auditors are assigned to engagements each consisting of various subtasks. All subtasks of an engagement have to be performed by the same auditor in prescribed time windows. The duration of subtasks differ depending on the performing auditor (Dodin *et al.*, 1998).

Besides the apparent analogies to resource constrained project scheduling, there is also a similarity to the vehicle routing problem with time windows. There, a fleet of vehicles serves a number of customers, even though not every vehicle has to be used. The problem is to assign customers to vehicles, and to generate a route for each of the vehicles, such that a performance criterion is optimal (Bramel and Simchi-Levi, 1997, Chapter 7).

Another related problem appears to be the assignment of non-preemptive computing tasks to groups of processors of a multi-processor system (Drozdowski, 1996). The size and number of groups of processors performing a set of tasks can vary, and time windows for the execution of tasks as well as precedence relations between computing tasks exist (Błażewicz and Liu, 1996). The term "gang scheduling" has been introduced in the context of multi-processor scheduling (Feitelson, 1996), but also relates to standard terms of port operations.

We model the logistics problem at hand as a multi-mode gang scheduling problem. Let $\mathcal{A}$ be the set of (non-preemptive) tasks involved in a problem. For each task $j \in \mathcal{A}$, a certain volume V_j is to be processed in a time interval specified by its earliest permissible starting time EST_j and its latest permissible finishing time LFT_j. The predecessor task of task j of an existing pairwise precedence relation is denoted by η_j. If no predecessor is defined, $\eta_j = \emptyset$.

Time is modelled by $1, \ldots, T$ discrete time steps, which are treated as periods rather than as points in time. If one task is complete at time t, its immediate successor task cannot start before $t + 1$.

The workers are grouped into a set of G gangs $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_G\}$. Each gang $\mathcal{G}_i$ is assigned a subset of the tasks $\mathcal{A}_i \subseteq \mathcal{A}$ with $\bigcup_{i=1}^{G} \mathcal{A}_i = \mathcal{A}$ and $\bigcap_{i=1}^{G} \mathcal{A}_i = \emptyset$. The number of workers in gang $\mathcal{G}_i$ is denoted by p_i, the number of tasks assigned to gang $\mathcal{G}_i$ is denoted by $h_i = |\mathcal{A}_i|$. At any time step, a gang can only work on a single task.

A solution is described by the number of workers in each gang, an assignment of tasks to gangs (i.e. a partition of $\mathcal{A}$ into $\mathcal{A}_i$), and a sequence of tasks for each gang.

Let the task on position k in gang i's permutation of tasks be denoted by $\pi_{i,k}$ (i.e. task $\pi_{i,k}$ with $k > 1$ is processed after task $\pi_{i,k-1}$). Starting times

of tasks $s_j \in [1,\ldots,T]$ can be derived from such a task sequence by assuming left-shifted scheduling at the earliest possible starting time. Similarly, the completion times $c_j \in [1,\ldots,T]$ of tasks are fully determined by the starting times and the manpower demand.

The model can be stated as follows:

$$\min \quad \sum_{i=1}^{G} p_i \tag{1}$$

$$s_j \geq EST_j, \qquad \forall j \in \mathcal{A} \tag{2}$$

$$c_j \leq LFT_j, \qquad \forall j \in \mathcal{A} \tag{3}$$

$$s_j > c_{\eta_j}, \qquad \forall j \in \mathcal{A} : \eta_j \neq \emptyset \tag{4}$$

$$s_{\pi_{i,j}} > c_{\pi_{i,j-1}}, \qquad \forall i \in \{1\ldots G\}, j \in \{2\ldots h_i\} \tag{5}$$

$$c_j = s_j + \left\lfloor \frac{V_j}{p_i} \right\rfloor, \qquad \forall i \in \{1\ldots G\}, \forall j \in \mathcal{A}_i \tag{6}$$

$$\mathcal{A}_i \subseteq \mathcal{A} \qquad \forall i \in \{1\ldots G\} \tag{7}$$

$$\bigcup_{i=1}^{G} \mathcal{A}_i = \mathcal{A} \tag{8}$$

$$\bigcap_{i=1}^{G} \mathcal{A}_i = \emptyset \tag{9}$$

Equation (1) minimises the sum of workers p_i over all gangs. Time windows of tasks are taken into account by (2) and (3). Precedence relations among tasks are considered by Eq. (4). Equation (5) ensures a feasible order of tasks belonging to the same gang. The completion time of each task is calculated in (6). Finally, Eqs. (7)–(9) make sure that each task is assigned to exactly one gang.

3. THE TABU SEARCH ALGORITHM

As already stated in the previous section, a solution has to specify three things:

1. The number of gangs G and the number of workers p_i for each gang $\mathcal{G}_i$
2. the assignment of tasks to gangs, and
3. the sequence of operations in each gang.

In this paper, we are going to use a tabu search algorithm to assign tasks to gangs. For every such assignment, inside the tabu search heuristic, an appropriate schedule for each gang is derived by applying a simple Schrage scheduler (Carlier, 1982).

The basic idea of tabu search is to iteratively move from a current solution s to another solution $s' \in \mathcal{N}(s)$ in the current solution's neighbourhood. The neighbourhood definition allows "small" changes to a solution, called moves, in order to navigate through the search space. The move to be executed is selected based on costs $C(s, s')$ associated with the move from s to s'. For minimisation problems, the deepest descent, mildest ascent strategy is applied for selecting moves.

In order to avoid cycling in a local optimum, recently performed moves are kept in a list of currently forbidden ("tabu") moves for a number of iterations. This tabu list is maintained and updated each time a move has been carried out (Glover and Laguna, 1993). We use a variable tabu list length.

In the remaining sections, the following four issues will be addressed in more detail:

- What is a suitable neighbourhood definition?
- How to build an initial solution?
- How to perform a move?
- How to estimate the cost of a move?

3.1 Neighbourhood and Tabu List

The purpose of the tabu search framework is the integration and disintegration of gangs by re-assigning tasks. We have chosen the most elementary move possible, namely the re-assignment of a single task from one gang to another, resulting in a neighbourhood size of roughly $H \cdot G$ with H being the number of tasks involved in a problem instance. We refrain from engaging more complex neighbourhood definitions like the exchange of two tasks between two gangs, because determining the costs $C(s, s')$ for all $s' \in \mathcal{N}(s)$ is computationally prohibitive and, as we will see, the remedy of estimating the costs becomes almost intractable for complex neighbourhoods.

As a move attribute we consider a task entering a gang. Consequently, a tabu list entry forbids a task to leave a certain gang for a certain number of iterations. Not every move is permissible. For example, for a certain interval of the planning horizon, whenever there are more tasks to be scheduled than there are time steps available, the move is excluded from $\mathcal{N}(s)$. Also, moves which disintegrate and integrate a gang at the same time are not considered.

3.2 Building an Initial Solution

The construction of a competitive and feasible solution is not trivial, because the necessary number of gangs is not known. We build an initial solution by separating tasks into as many gangs as possible. Tasks without precedence

relations are placed in a gang of their own, whereas pairs of tasks coupled by a precedence relation are processed by the same gang. In this way, precedence relations can be handled within the local scope of individual gangs, and it is guaranteed that the initial solution is feasible.

For each gang, we determine the minimum number of workers required to process the tasks. For each task j, the number of workers required is $r_j = \lceil V_j/((c_j - s_j) + 1)\rceil$.

In the event that only one task is assigned to a gang i, its minimum number of workers is thus $p_i = \lceil V_j/((LFT_j - EST_j) + 1)\rceil$

If two tasks j and k with $\eta_k = j$ share a gang, the number of workers required is at least as high as the maximum required for each task separately, and at least as high as if the two tasks would be treated as one:

$$p_i \geq \max_{l \in \{j,k\}} \lceil V_l/((LFT_l - EST_l) + 1)\rceil \quad (10)$$

$$p_i \geq \lceil (V_j + V_k)/((LFT_k - EST_j) + 1)\rceil \quad (11)$$

We start with the maximum of these two lower bounds and check for feasibility. If the number of workers is not sufficient to ensure feasibility, we iteratively increase the number of workers by 1 until the schedule becomes feasible.

3.3 Performing a Move

A move is performed in three steps: first, we move the task from one gang to the other. Then, these two gangs are re-scheduled, and contingently their mode (number of workers) is adapted. Finally, it is checked whether the re-scheduling of the two directly involved gangs can also lead to improvements in other gangs due to dynamically changed time windows. These aspects shall be discussed in the following.

Scheduling a Single Gang Let us first assume that the number of workers is given. This problem can be seen as a one-machine scheduling problem with heads and tails, where the head of a task denotes the non-available interval from $t = 1$ to EST_j, and the tail denotes the corresponding interval from $t = LFT_j$ up to the planning horizon T. The head of task no. 5 in Figure 1 ranges from time unit 1 to 8, whereas its tail comprises only time unit 18. Consequently, the time window of task no. 5 covers time units 9–17. In the current mode of processing, two time units are covered.

For our purpose, we extend the notion of heads and tails by the consideration of precedence relations of tasks placed in different gangs. Since only one gang is modified at a time, predecessor and successor tasks placed in other gangs may additionally constrain the temporal placement of tasks. The functions $est()$ and $lft()$ restrict the time window of a task to its currently largest

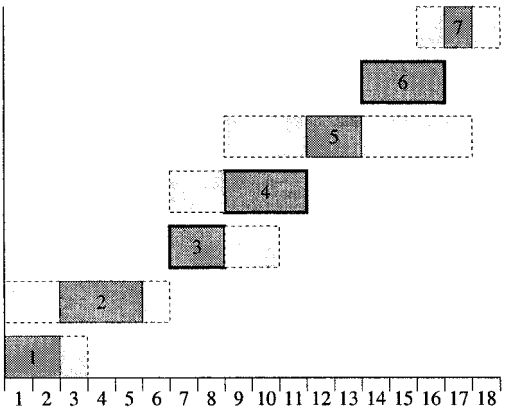


Figure 1. Example of Schrage schedule consisting of seven tasks to be scheduled in 18 time units. Dark grey rectangles represent the time of processing while light grey rectangles depict the time windows given. Critical tasks are indicated by a black border.

permissible extension. $est(j)$ returns EST_j if no predecessor η_j exists and $\max\{EST_j, c_{\eta_j} + 1\}$ otherwise. Similarly, $lft(j)$ returns LFT_j if no successor task κ_j exists and $\min\{LFT_j, s_{\kappa_j} - 1\}$ otherwise.

For the objective of minimising the makespan, this problem has been shown to be NP-hard (Carlier, 1982). Carlier proposes a branch & bound algorithm, which alters a schedule built by the Schrage heuristic and solves the problem optimally. However, because we have to schedule many gangs for each tabu search move, we have to rely on a very fast heuristic. We therefore rely on a single run of the Schrage heuristic which schedules all tasks of $\mathcal{A}_i$ in the planning horizon $1, \ldots, T$ in a given mode p_i.

Basically, the Schrage heuristic $schrage()$ schedules tasks sequentially with respect to the smallest permissible finishing time. In every iteration, one task is placed starting at time unit t. For this purpose, all tasks with $est() \leq t$ enter the selection set $\mathcal{S}$. If $|\mathcal{S}| = 0$, then t is increased to the minimum $est()$ of all tasks not yet scheduled. Otherwise, from $\mathcal{S}$, the task j with the smallest $lft()$ is selected. If it can be placed at time unit t in mode p_i without violating its time window, starting and completion time of j are determined, t is updated and finally j is removed from further consideration.

Figure 1 shows a schedule built by the Schrage heuristic. Initially, tasks no. 1 and 2 can be scheduled at $t = 1$. Task no. 1 is given preference because of its smaller $lft(1) = 3$. In the second iteration only task no. 2 can be placed at $t = 3$. In iteration three, no task is available at $t = 6$ and therefore t is updated to the minimal starting time of the remaining tasks $t = 7$. Task no. 3 dominates no. 4 due to its smaller $lft()$. Then, no. 4 is placed at its latest

permissible time of placement. Finally, the placement of tasks no. 5, 6, and 7 complete the schedule.

The number of workers required is determined in a similar way as for the initial solution described in the previous section. First, let us generalise the calculation of a lower bound to an arbitrary set of tasks $\mathcal{A}_i$. The number of workers required is at least as high as the maximum of the numbers required for any single task, assuming each task can utilise its entire time window (p'). Also, it is at least as high as if the set of tasks is treated as one task (p'').

More formally, the lower bound calculation looks as follows:

function $lower\_bound(\mathcal{A}_i)$
  for all $j \in \mathcal{A}_i$ do $r_j = \lceil V_j/((lft_j - est_j) + 1)\rceil$
  $p' = \max_{j\in\mathcal{A}_i}\{r_j\}$
  $u = \sum_{t=1}^{T} usable(t, \mathcal{A}_i)$
  $p'' = \lceil(\sum_{j\in\mathcal{A}_i} V_j)/u\rceil$
  return $\max\{p', p''\}$
end function

where function $usable()$ returns 1 if time step t can be utilised by at least one task, and 0 otherwise.

As for the initial solution, we first set $p_i = lower\_bound(\mathcal{A}_i)$ and contingently increase p_i by one as long as the Schrage heuristic fails to place all tasks without violating a time window.

procedure $schedule(\mathcal{A}_i)$
  $p_i = lower\_bound(\mathcal{A}_i)$
  while $schrage(\mathcal{A}_i, p_i) =$ `false` do
    $p_i = p_i + 1$
  end while
end procedure

Propagation of Time Windows Scheduling the tasks of a gang may also entail the re-scheduling of other gangs. In the event that tasks of gang i and k involved in the move have precedence constraints with tasks of other gangs, a change of i's or k's schedule may change the time windows of tasks in other gangs, which might lead to different (better or worse) schedules if the Schrage heuristic were applied again.

In particular, we exploit new opportunities due to an enlargement of a time window in the event that the completion time c_j of task j impedes an earlier starting time s_l of the successor task l in another gang. Thus, whenever $c_j+1 = s_l$ holds, and c_j decreases because the gang of task j is re-scheduled, also the gang of task l is noted for re-scheduling. Similarly, whenever the starting time s_l of task l has impeded a later completion at c_j of the predecessor task j

in another gang, that gang is noted for re-scheduling. Once noted, gangs are re-scheduled in a random order.

Since time windows can be recursively extended, we call this procedure *time-window propagation.* The prerequisites for propagating a time window are rarely satisfied, such that the number of re-scheduling activities triggered is limited. If, however, gang i is noted for re-scheduling, there is a reasonable chance to decrease the number of workers p_i.

3.4 Estimating the Cost of a Move

Since the simulation of a move implies at least two, but often many more calls to the Schrage heuristic, it would be computationally very burdensome. Therefore, we estimate a move's effects on the two gangs directly involved and neglect further effects of the propagation of time windows.

To estimate the costs of a move we determine a contingent savings of workers $p_i - \hat{p}_i$ due to the removal of a task j from gang i. Next, we determine the additional effort $\hat{p}_k - p_k$ spent on integrating j into another gang k. We calculate the difference of the two figures, i.e. $\hat{p}_i + \hat{p}_k - p_i - p_k$, and select the move with the highest approximated gain (or the lowest approximated loss) for execution.

Schedule Properties In order to estimate $\hat{p}_i$ and $\hat{p}_k$ for gangs i and k, we discuss some properties of schedules which will help to derive appropriate estimates. Central notions of our arguments are the *block* of tasks and the *criticality* of tasks.

Definition 1 A *block* consists of a sequence of tasks processed without interruption, where the first task starts at its earliest possible starting time, and all other tasks start later than their earliest possible starting time.

Tasks of a block are placed by the Schrage heuristic independent of all other tasks not belonging to this block. Therefore, blocks separate a schedule into several parts, which can be considered independently. Another interesting property concerning blocks is that slack can occur only at the end of blocks or before the first block.

In Figure 1 we identify three blocks. Block 1 consists of tasks no. 1 and 2 and is easily distinguished from block 2 consisting of tasks no. 3, 4, and 5 by the idle-time of time unit 6. Block 3 consisting of tasks no. 6 and 7 can be identified by considering $EST_6 = s_6 = 14$.

Definition 2 A task is said to be *critical* if it is involved in a sequence of tasks (called a critical path), which cannot be shifted back or forth in any way without increasing the number of workers involved.

In a Schrage-schedule, a critical block causes the violation of a time-window constraint if the number of workers is decreased by one. Obviously, all tasks of a critical path belong to the same block, but not every block necessarily contains a critical path. However, if a critical path exists, it starts with the first task of a block. A critical path terminates with the last critical task of its block. Thus, it completes processing at its latest finishing time, although there may exist additional non-critical tasks scheduled later in the same block.

Only one critical path can exist in a block. In the event that a task j scheduled directly before a critical task k causes k's criticality, obviously j itself must be critical. Accordingly, if we classify any task to be critical, all preceding tasks of its block are critical too.

In Figure 1 none of the tasks no. 1 and 2 forming the first block are critical, because the entire block could be shifted to the right by one time unit without delaying other tasks. Tasks 3 and 4 form a critical path within the second block. Although task no. 5 cannot be shifted, it is not critical by definition, because it does not complete at its latest finishing time. Task no. 6 is again critical without the participation of other tasks placed.

As we will see, the notions of blocks and critical tasks make a valuable contribution to the estimation of a move.

Estimating the Manpower Release of a Task Removal Obviously, every schedule has at least one critical block (a block containing a critical path), which impedes a further decrease of the number of workers p_i. For that reason, the only way to obtain a benefit from removing a task from a gang is to break a critical block. If two or more critical blocks exist, and one critical block breaks, at least one other critical block remains unchanged and consequently no benefit can be gained. For instance, in Figure 1 the removal of the block consisting of task no. 6 cannot lead to an improvement because tasks no. 3 and 4 still remain critical. If only one critical block exists, but a non-critical task is removed, again no saving can be expected. In all these cases the estimation procedure returns p_i.

Estimating the Manpower Demand of the Critical Block Removing a critical task from the only critical block existing can lead to a decrease of the number of workers involved. Here we have to distinguish the removal of a task (a) within a critical path, (b) at the beginning of a critical path, and finally, (c) at the end of a critical path.

(a) In case of the removal of a task inside a path we determine the time-span stretching from the starting time s_j of the first task j of the block to the completion time c_l of the last critical task l of the block. We sum the volumes of all tasks but the one to be removed from the critical path, and divide the sum of volumes through the extension of the time-

span. Consider a critical path consisting of tasks 1, 2 and 3. If task 2 is removed, the new number of workers is estimated by $(V_1 + V_3)/((c_3 - s_1) + 1)$.

(b) The removal of the first task j of a critical path alters the starting condition of the path. Therefore the path can start at the maximum of the earliest possible starting time est_l of the second task l of the path, and the completion time $c_m + 1$ of the predecessor task m of task j to be removed (if m does not exists, set $c_m = 0$). For the example of a critical path consisting of tasks 1, 2 and 3, the new number of workers after removing task 1 is estimated by $(V_2 + V_3)/((c_3 - c_{m_1}) + 1)$.

(c) Similarly, the removal of the last task of a critical path alters the terminating condition of the path. Therefore the path can complete at the minimum of the latest possible completion time lft_j of the last but one task j of the path, and the starting time $s_l - 1$ of task l succeeding the path (if l does not exist, set $s_l = T + 1$).

Integrating the Bound Imposed by Non-critical Blocks The approximated manpower demand refers to one critical block only. After the removal of a critical task from this block, other, so far non-critical, blocks may become critical, and for that reason may limit a further decrease of the manpower demand.

Consider a critical block b_c for which the removal of a critical task has been estimated. For blocks in $\mathcal{B} = \{b_1, \ldots, b_{c-1}, b_{c+1}, \ldots, b_n\}$ a lower bound on the number of workers required is calculated by prorating the sum of volumes of its tasks onto the time-span used by the block plus a contingent idle-time following the block. The number of workers applicable is then approximated by the workers $\hat{p}_c$ determined for block c and for the other blocks of $\mathcal{B}$ by $\hat{p}_i = \max\{\hat{p}_c, \max_{k \in \mathcal{B}}\{\hat{p}_k\}\}$.

The procedure accurately identifies the vast majority of non-improving task removals. If the removal of critical tasks in the only critical block existing may lead to a benefit, this benefit is limited by the manpower capacity required for other blocks. In all these cases a conservative estimate $\hat{p}_i$ is returned by the procedure.

Estimating the Manpower Demand of a Task Insertion For estimating the effect on p_k caused by the insertion of a task v into gang k, we first try to fit this task into an existing gang schedule. If v integrates a new gang, we calculate the exact manpower demand p_k of the new gang k as $p_k = \lceil V_j/((lft_j - est_j) + 1)\rceil$. In all other cases, we estimate the additional number of workers $(\hat{p}_k - p_k)$ required in order to produce a feasible schedule.

We start by determining the task w in an existing gang schedule, before which the new task v will be inserted. To identify w, we scan the tasks of the schedule in the order produced by the Schrage heuristic and stop at the first task w with $est_v \leq est_w$ and $lft_v < lft_w$. After having found w, we determine the earliest permissible starting time s_v and the latest permissible completion time c_v with respect to the existing gang structure.

If v has to be appended to the end of the schedule, we easily check whether contingent idle-time $T - c_j$ after the last task j suffices to integrate v.

If v is to be inserted, we are going to verify the available idle-time. Idle-time to the left of w can be available only if w is the first operation of a block. In this case v may start right after the completion time of w's predecessor u, i.e. at time step $c_u + 1$. The utilisation of the idle-time, however, is limited by est_v, thus $s_v = \max\{c_u + 1, est_v\}$.

Idle-time on the right can be available only if w is non-critical. In this case w and its non-critical successor tasks can be shifted to the right in order to obtain additional idle-time. The maximal amount of idle-time available can be determined by placing the tasks right-shifted in the opposite order of the task sequence given in the Schrage schedule. We refer to the resulting starting time of task w as $\overline{s}_w$. Thus, $c_v = \min\{\overline{s}_w - 1, lft_v\}$.

In the event that $\lceil V_v/((c_v - s_v) + 1)\rceil$ is smaller than or equal to the number of workers currently engaged, task v can be integrated in the schedule without engaging additional workers. The procedure returns $\hat{p}_i = p_i$ and terminates.

Whenever the number of workers does not suffice to integrate task v, the additional manpower demand has to be estimated. Since task v for sure becomes critical, the blocks merged by v are considered for prorating v's volume by means of function *lower_bound*(). The approximated increase of the number of workers $\hat{p}_i$ is returned to the calling procedure.

Summary of the Approximation Effort Summarising, the estimation scheme proposed accurately identifies the vast majority of task removals which do not yield savings of workers. In other cases the estimation tends to produce a conservative estimate of savings to be gained. Apart from some special cases, only those insertions are estimated correctly which do not require additional workers. In other cases again a conservative estimate on the additional number of workers required is determined. Whether the approximation quality suffices in order to guide the search successfully will be examined in the following section.

4. COMPUTATIONAL INVESTIGATION

To assess the performance of our tabu search algorithm, we will evaluate it empirically. To that end, we will first propose a benchmark generator, and then

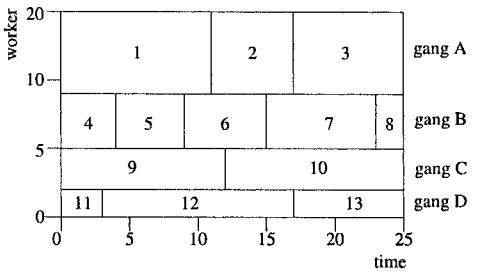


Figure 2. Exemplary optimal solution to a problem instance.

compare our tabu search algorithm with a simple local search approach on a variety of benchmark problems.

4.1 Generating Problem Instances

In this section, we present a way to produce benchmark instances for which the optima are known. The basic idea is to view a schedule as a rectangular plane, where the horizontal extension represents the time dimension and the vertical extension denotes the number of workers involved. Obviously, an entirely filled rectangle means that all workers are occupied for the whole planning horizon, i.e. an optimal schedule.

The example in Figure 2 shows a schedule with 13 tasks organised in 4 gangs. The planning horizon comprises of 25 time units for which 20 workers are utilised. Time windows and precedence relations are not yet specified, but can be easily added to the schedule.

A benchmark instance is generated based on the following input parameters:

- the number of time units in the planning horizon T
- the total number of tasks H
- the number of gangs G.
- the minimal and maximal number of workers in a gang $p_{\min}$ resp. $p_{\max}$
- the percentage of tasks involved in a precedence relation $\gamma \in [0, 1]$
- a parameter $\omega \in [0, 1]$ determining the extension of time windows.

To produce a benchmark instance we proceed as follows:

1. The number of tasks h_i for gang i is determined by prorating the H tasks uniformly upon the G gangs. This can be implemented by first initialising array K of dimension $[0, G]$ and setting $K[0] = 0$ and $K[G] = H$.

Table 1. The mean relative error of the local optima found by the local hill-climber.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	2.5	6.3	10.4	18.0	2.1	5.9	10.4	17.3
0.25	2.6	6.5	10.6	16.8	2.1	6.0	10.2	16.2
0.50	2.7	8.1	11.9	17.2	2.3	6.5	10.9	15.9
0.75	3.3	9.9	14.6	18.6	3.1	9.1	13.6	17.1
1.00	5.8	13.3	17.1	20.5	6.0	12.8	16.8	18.7

Then, we assign uniformly distributed random numbers in the range $[1, H-1]$ to $K[1], \ldots, K[G-1]$. Next, we sort K in ascending order, and finally we determine $h_i := K[i] - K[i-1]$.

2. The starting times s_j of task $j \in \mathcal{A}_i$ in gang i are determined by distributing the h_i tasks onto the T time units. We proceed in analogy to the procedure described for Step 1. The completion times c_j of tasks are determined by means of the gang's task chain: $c_{\eta_j} := s_j - 1$.

3. The number of workers p_i of gang i is drawn uniformly distributed from the range $[p_{\min}, p_{\max}]$. Finally we calculate the task volume by multiplying the task's duration with its manpower demand, i.e. $V_j := ((c_j - s_j) + 1) \cdot p_i$.

4. A task can have a precedence relation with every other non-overlapping task of the schedule. For example, in Figure 2 task no. 2 can have a precedence relation with tasks in $\{1, 3, 4, 5, 8, 11, 13\}$. We iteratively select random non-overlapping pairs of tasks not yet involved in a precedence relation and insert a precedence relation until a fraction γ of the tasks are involved in a precedence relation or there is no pair of non-overlapping tasks left.

5. Finally time windows are specified in percent ω of the examined time horizon. In particular, we determine $EST_j := \lceil s_j \cdot \omega \rceil$ and $LFT_j := \lceil c_j + (T - c_j) \cdot \omega \rceil$.

4.2 Empirical Results

To assess the potential of our tabu search procedure, we generate problem instances in two sizes, namely with 10 gangs and 100 tasks (small problems) and with 20 gangs and 200 tasks (large problems). For each size, we additionally vary the extension of time windows ω and the percentage of tasks

Table 2. Mean number of hill-climbing moves performed.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	37.4	52.5	59.2	65.9	80.3	104.8	118.5	133.9
0.25	31.7	45.7	51.4	59.3	70.9	91.1	103.1	120.2
0.50	26.0	36.4	42.5	52.7	58.1	78.1	88.1	110.0
0.75	18.8	25.2	33.9	50.2	42.7	54.2	72.9	104.8
1.00	3.8	11.7	28.3	47.7	10.5	27.3	55.8	99.0

involved in precedence relations, γ. Time windows are generated with $\omega \in \{0.25, 0.50, 0.75, 1.00\}$, and the percentage of tasks coupled by a precedence relation are given by $\gamma \in \{0.0, 0.25, 0.50, 0.75, 1.00\}$.

As parameters for the tabu search algorithm, we use a variable tabu list length of $[5, \sqrt{H}]$ where H is the number of tasks involved in the problem, and the stopping criterion is fixed at 10,000 iterations.

For every combination of size, time-window extension and number of precedence relations specified, 10 benchmark instances are generated which are solved three times each, because of the non-deterministic nature of the variable sized tabu list length used. Overall, every figure given in Tables 3 and 4 represents the mean over 30 samples observed.

In order to gauge the competitiveness of the tabu search procedure, additionally a local hill-climber is applied. The algorithm starts from the same initial solution as proposed in Section 3.2 and uses the same neighbourhood definition as the tabu search procedure, refer to Section 3.1. Different from tabu search, the local hill-climber calculates its $C(s, s')$ exactly by simulating all moves in advance. Iteratively, the move yielding the greatest reduction in the number of workers is performed until a local optimum is reached.

For the local hill-climber, the mean relative error (against the optimal solution) over the 10 benchmark instances is presented in Table 1. For unconstrained problems, the relative error is quite small, with 2.5% and 2.1% for small and large instances respectively. However, with an increasing tightness of the constraints imposed, the relative error increases drastically to approximately 20% for $\gamma = 1.00$ and $\omega = 0.25$.

Obviously, the search space becomes more rugged, which goes along with a decreasing performance of the hill-climber. However, as shown in Table 2, the number of hill-climbing moves performed does not directly reflect the ruggedness of the space to be searched. As one may expect, the hill-climbing paths get shorter with an increasing number of precedence constraints imposed (γ). The reasonable relative error of $\approx$6% for $\gamma = 1.0$ and $\omega = 1.0$ is obtained by

Table 3. The mean relative error of the best solutions found by the tabu search procedure.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	4.59	8.86	6.87	8.67	4.71	8.76	6.89	8.77
0.25	4.83	8.46	7.42	8.36	4.84	8.20	8.21	9.60
0.50	4.39	7.27	8.15	9.45	5.04	9.47	8.81	9.65
0.75	4.32	7.71	9.93	9.81	4.95	10.32	9.97	10.70
1.00	3.69	9.64	9.19	10.01	4.62	9.16	9.30	10.92

a mere 3.8 moves on average for small problems and 10.5 for large problems respectively.

By narrowing the time windows starting from the entire planning horizon ($\omega = 1.0$) towards 1/4th of the horizon ($\omega = 0.25$), the number of moves performed on a downhill walk increases significantly. Apparently, tight time windows introduce a locality to search such that only tiny improvements per move can be obtained. Although with $\omega = 0.25$ more than 100 moves are performed for large problems, the relative error obtained increases with an increasing tightness of time windows.

Despite performing an increasing number of moves, an increasing relative error is observed. Thus, a further improvement in searching a rugged search space requires the temporary deterioration of the objective function value. Although the tabu search procedure provides this feature, the large number of iterations needed requires a considerably faster estimation of move costs.

Table 3 presents the mean relative error observed for the tabu search procedure. For a maximal time-window extension $\omega = 1.00$ the relative error comprises $\approx$4% regardless of the number of precedence relations specified. This is twice the relative error observed for the hill-climber, and pinpoints at the shortcoming of the estimation procedure. Obviously, the estimation delivers poor approximation of the changes in the number of workers imposed by a move in the case of loose constraints.

Narrowing the time windows increases the relative error of the tabu search procedure only slightly up to $\approx$10% for $\omega = 0.25$. This figure is approximately half of the relative error observed for the hill-climber, which convincingly demonstrates the advantage of tabu search for problem instances with tight constraints.

The tighter the constraints are, the better the costs of a move are approximated by our estimation procedure, because the time span onto which processing times are prorated decreases with an increasing tightness of constraints. Therefore, the estimation procedure is able to guide the search more accu-

Table 4. Number of gangs recorded with the best solution observed.

	Small problems				Large problems			
γ/ω	1.00	0.75	0.50	0.25	1.00	0.75	0.50	0.25
0.00	70.2	30.5	16.2	11.2	138.5	61.3	29.3	20.5
0.25	64.6	29.1	14.1	11.0	125.5	51.3	23.9	19.4
0.50	57.1	20.5	14.6	11.3	115.3	54.4	27.7	19.0
0.75	49.8	18.7	14.9	10.8	100.7	56.2	25.8	18.3
1.00	39.5	28.8	13.3	10.5	82.9	36.1	21.4	18.4

rately. The algorithm proposed seems pleasantly robust against the existence of precedence relations and an increasing problem size.

Table 4 shows the mean number of gangs recorded with the best solution observed. This figure demonstrates how well the gang structure of the optimal solution has been reproduced by the tabu search procedure. For large time windows ($\omega = 1.00$) an enormous number of gangs have been integrated. Since only the integer condition on the number of workers restricts the algorithm from finding the optimal solution, there is no need to reduce the number of gangs. However, merely the existence of precedence relations (which has only a small influence on the solution quality) cuts the number of gangs in half.

For higher constrained problems with $\gamma \geq 0.5$ and $\omega \leq 0.5$ the gang structure of the optimal solution is successfully approximated. Here, for small problem instances $\approx$10 gangs are integrated, whereas the 200 tasks of large problem instances are essentially distributed among $\approx$20 gangs. For highly constrained problems an effective gang structure is a prerequisite for obtaining high quality solutions. Obviously, this structure is identified by the algorithmic approach proposed.

5. CONCLUSION

In this paper, we have addressed the problem of finding a suitable gang structure for a task scheduling problem. For this problem, we have presented a model and we have proposed a way to generate benchmark instances of varying properties. We have developed an efficient tabu search procedure in order to solve such gang scheduling problems.

Particular attention has been paid to the design of the Schrage-scheduler acting as a base-heuristic in the tabu search framework. Although the move of a task from one gang to another modifies just these two gangs directly, the other gangs are indirectly affected by the change of time-window constraints and have to be rescheduled as well.

Since the move neighbourhood is large and the calculation of the cost or benefit is computationally expensive, we have proposed a cost estimation procedure, which approximates the outcome of a move before it is actually performed. Although an estimate must be imperfect in the face of the move's complexity, experience has confirmed the applicability of the estimate developed. For a wide range of benchmark instances a promising solution quality has been achieved.

References

Błażewicz, J. and Liu, Z. (1996) Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research*, **94**:231–241.

Bramel, J. and Simchi-Levi, D. (1997) *The Logic of Logistics*. Operations Research Series. Springer, Berlin.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, **112**:3–41.

Carlier, J. (1982) The one-machine scheduling problem. *European Journal of Operational Research*, **11**:42–47.

Dodin, B., Elimam, A. A., and Rolland, E. (1998) Tabu search in audit scheduling. *European Journal of Operational Research*, **106**:373–392.

Drozdowski, M. (1996) Scheduling multiprocessor tasks—an overview. *European Journal of Operational Research*, **94**:215–230.

Feitelson, D. G. (1996) Packing schemes for gang scheduling. In Feitelson, D. G. and Rudolph, L. (Eds.), *Job Scheduing Strategies for Parallel Processing*, Lecture Notes in Computer Science, Vol. 1162, Springer, Berlin, pp. 89–110.

Glover, F. and Laguna, M. (1993) Tabu search. In Reeves, Colin R. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, Oxford, pp. 70–150.

Mattfeld, D. C. and Kopfer, H. (2003) Terminal operations management in vehicle transshipment. *Transportation Research A*, 37.

Salewski, F., Schirmer, A., and Drexl, A. (1997) Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. *European Journal of Operational Research*, **102**:88–110.

Scheduling in Space

CONSTRAINT-BASED RANDOM SEARCH FOR SOLVING SPACECRAFT DOWNLINK SCHEDULING PROBLEMS

Angelo Oddi, Nicola Policella, Amedeo Cesta, and Gabriella Cortellessa
Planning and Scheduling Team
ISTC-CNR Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy
{a.oddi, policella, a.cesta, corte}@istc.cnr.it

Abstract This paper introduces a combinatorial optimisation problem called the MARS EXPRESS Memory Dumping Problem (MEX-MDP), which arises in the European Space Agency programme MARS EXPRESS. The domain is characterised by complex constraints concerning bounded on-board memory capacities, limited communication windows over the downlink channels, deadlines and ready times imposed by the scientists using the spacecraft instruments. This paper lays out the problem and analyses its computational complexity showing that MEX-MDP is *NP-hard*. Then the problem is modelled as a Constraint Satisfaction Problem and two different heuristic strategies for its solution are presented: a core greedy constraint-based procedure and an iterative sampling strategy based on *random search*. The algorithms are evaluated both against a benchmark set created on the basis of ESA documentation and a *lower bound* of the minimised objective function. Experimental results show the overall effectiveness of the approach.

Keywords: constraint reasoning, random sampling, greedy heuristic search.

1. INTRODUCTION

MARS-EXPRESS is a European Space Agency (ESA) programme that launched a spacecraft toward Mars on June 2, 2003. The space probe has been orbiting around the Red Planet since the beginning of 2004 for two years operating seven different payloads. MARS-EXPRESS represents a challenging and interesting domain for research in automated problem solving. The support of a complete mission planning activity is a challenging goal involving several sub-activities. The dumping of the on-board memories to the ground station is one such sub-activity that the authors have tackled in a study within the MARS-EXPRESS programme. The problem has been formalised as the

MARS-EXPRESS Memory Dumping Problem (MEX-MDP), an optimisation problem that involves the sequencing of dumping operations under complex constraints such as maximal data rate in communication links, limited communication windows, operation ready times and bounded on-board packet store capacities.

Problems similar to MEX-MDP arise in satellite domains such as the ones described in Verfaillie and Lemaitre (2001) and Bensana *et al.* (1999). These works concern a set of Earth observation operations to be allocated over time under a set of mandatory constraints such as no overlapping images, sufficient transition times (or *setup* times), bounded instantaneous data flow and on-board limited recording capacity. The problems are addressed with a variety of solving techniques ranging from Integer Linear Programming, Constraint Programming, Branch&Bound and Local Search techniques like *tabu search*. The papers show that the best solutions are not always obtained with a single solving technique. Different needs, and different trade-offs between quality and computation time, are addressed by different algorithms or combinations of solving techniques through a *meta-heuristic* schema.

A similar approach is followed in this paper, which proposes a meta-heuristic strategy based on the integration of Random Search (Motwani and Raghavan, 1995; Resende and Riberio, 2002) and Constraint Satisfaction Problem (CSP) solving. It turns out that such combination, also used to approach other scheduling problems (i.e. Nuijten and Aarts, 1996; Oddi and Smith, 1997; Cesta *et al.*, 2002b), is effective to tackle the MEX-MDP problem.

The paper is structured as follows. Section 2 introduces the domain of work and the modelling process followed to define MEX-MDP. In addition, the problem's computational complexity is studied and the problem is shown to be *NP-hard*. Section 3 describes a formalisation of the problem as a CSP while Section 4 presents the solving algorithms. Section 5 introduces a set of benchmarks for MEX-MDP, computes a *lower bound* for the problem instances and analyses the experimental results. A discussion and some concluding remarks close the paper.

2. THE MEMORY DUMPING PROBLEM

In a deep-space mission like MARS-EXPRESS, data transmission to Earth is a key issue. The space probe continuously produces a large amount of data which derives from the activities of its payloads (the on-board scientific instruments) and from on-board device monitoring and verification tasks (the so-called *housekeeping data*). All this data, usually referred to as *telemetry*, is to be transferred to Earth during downlink time intervals (the temporal windows in which the spacecraft points to Earth). MARS-EXPRESS is endowed with a single pointing system, thus during regular operations it will either point

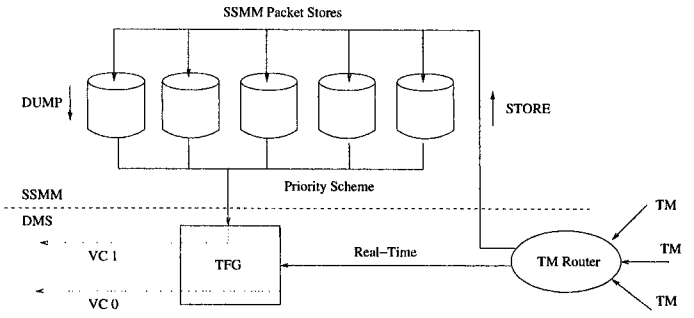


Figure 1. On-board telemetry flow.

to Mars and perform payload operations or point to Earth and transmit data through the downlink channel. As a consequence, data are first stored on the on-board solid state mass memory and then transferred to Earth. The problem solving goal consists in synthesising spacecraft operations for emptying the on-board memory during downlink time intervals, so allowing the spacecraft to save new information without losing previous data. A further goal is to minimise a given objective function. For example, the turnover time—or flow-time—of a single scientific operations is the time interval from the end of data production on board and its availability on Earth. An objective function used in practice is the average turnover time over a set of scientific operations.

Several constraints conflict with the problem solving goal. Besides the just-described communication channel availability, different transmission rates are to be taken into account during different communication intervals. Additionally, constraints arise from the specific use of the on-board memory which is subdivided into different memory banks (or packet stores) with a finite capacity. For each piece of information produced inside the probe, one or more packet stores are defined in which such data are stored. Each packet store can be seen as a file of a given maximal size where data are inserted with a cyclic policy—so previous information is overwritten if the amount of data overflows the packet store capacity. Overwriting has to be maximally avoided because it implies loss of data and consequent failure of certain observation programs.

We have formalised this problem as the Mars Express Memory Dumping Problem (MEX-MDP) whose different features are described in the rest of this section.

2.1 Basic Modelling

Figure 1 schematically illustrates the parts of the spacecraft that are relevant for the MEX-MDP. The figure shows the different telemetry (TM) data

produced on-board and then stored in the solid state mass memory (SSMM) that is subdivided into packet stores. Data accumulated by memory stores are downloaded to Earth with different dumps.

The basic objects that are relevant to the MEX-MDP domain can be subdivided into either *resources* or *activities*. *Resources* represent domain subsystems able to give services, while *activities* model tasks to be executed using resources over time.

Resources used in our model of MARS-EXPRESS are:

- *Solid State Mass Memory (SSMM).* The SSMM is able to store both science and housekeeping (HK) information. SSMM is subdivided into a set of *packet stores* $\{pks_1, pks_2, \ldots, pks_m\}$, each one with a fixed capacity c_i and a priority p_i for dumping data.

- *Communication Channels.* The downlink connections to Earth for transmitting data. These resources are characterised by a set of separated communication windows $CW = \{w_i\}$ that identify time intervals for downlink. Each element w_i is a 3-tuple $\langle dr_i, s_i, e_i \rangle$, where dr_i is the available data rate during the time window w_i and s_i and e_i are respectively the start and the end time of w_i.

It is worth noting that data are stored or downlinked according to a further subdivision into *data packets* $\{pk_j\}$. A data packet is a set of sequential data with a header containing information like size (a constant value for each packet), application ID, sequence source counter, time, flags, etc. A discrete model of time has been used such that all the events happen within an integer interval of time $[0, H]$ (time horizon). Moreover, all data quantities (data rates) have been expressed in *data units* (*data units per second*), where a data unit coincides with the size of a single data packet.

Activities describe *how* resources can be used. Each activity a_i is characterised by a fixed duration d_i and two variables s_i and e_i which respectively represent its start time and end time. Two basic types are relevant to MEX-MDP: store operations st_i and memory dumps md_i.

- *Store Operation.* Each st_i "instantaneously" stores an amount of data q_i at its end time in a destination packet store pks_j.

- *Memory Dump.* An activity that transmits a set of data packets from a packet store to the ground station. It models a telecommand executed by the spacecraft. A memory dump operation md_i transfers a set of data

packets $\{pk_j\}$ from a packet store to a transfer device (the transfer frame generator (TFG) shown in Figure 1).

In the MEX-MDP domain there are two different sources of store operations st_i: the *Payload Operation Request* (por_i) and a set of sources of housekeeping data, which can be seen as a continuous stream of data with a given flat rate; we call these sources *Continuous Data Stream* (CDS).

A *Payload Operation Request* is a model for a scientific observation which generates a set of data distributed over a subset of the available packet stores. According to this model, the produced data are decomposed in a set of different *store* operations such that, $por_i = \{st_{ij}\}$, all of them with the same durations and start times.

On the other hand, a *Continuous Data Stream* models an on-board process which works in "background" with respect to the scientific activities. It generates a flow of data with constant rate which has to be stored in the SSMM. Examples of data streams are the housekeeping data collected on a regular basis to control the behaviour of the on-board sub-systems.

These are different types of data sources. In fact, a por_i is a time bounded activity, which stores data at its end time, whereas a CDS is a continuous data flow over the domain horizon. However, we choose to also model a CDS as a periodic sequence of store operations. In particular, given a CDS with a flat rate r, we define a period T_{cds}, such that, for each instant of time $t_j = j \cdot T_{cds}$ ($j = 1, 2, \dots$) an activity st_{ij} stores an amount of data equal to $r \cdot T_{cds}$. Hence, we can consider as input data for the problem just an equivalent set of store operations containing data packets, such that each packet contains a pointer to its source.

2.2 Problem Definition

Given these basic domain entities, let us now define the MEX-MDP. A set of scientific observations, $\mathcal{POR} = \{por_1, por_2, \dots, por_n\}$ and a set of housekeeping productions, $\mathcal{CDS} = \{cds_1, cds_2, \dots, cds_m\}$, are both reduced to a set of store operations on the on-board memory. A *solution* to a MEX-MDP is a set of dumping operations $S = \{md_1, md_2, \dots, md_s\}$ such that

- the whole set of data are "available" on ground within the considered temporal horizon $\mathcal{H} = [0, H]$.
- Each dump operation starts after the generation of the corresponding data. For each packet store, the data are moved through the communication channel according to a First In First Out (FIFO) policy.
- Each dump activity, md_i, is executed within an assigned time window w_j which has a constant data rate r_j. Moreover, dump operations cannot reciprocally overlap.

- At each instant $t \in \mathcal{H}$, the amount of data stored in each packet store pks_i has to be less or equal to the packet store capacity c_i (i.e. overwriting is not allowed).

The additional goal is to find *high-quality solutions* with respect to a set of evaluation parameters: a high-quality plan delivers all the stored data as soon as possible according to a definite policy or objective function. A relevant piece of information to define an objective function is the *turnover time* of a payload operation por_i:

$$tt(por_i) = del(por_i) - e(por_i)$$

where $del(por_i)$ is the delivery time of por_i and $e(por_i)$ is its end time. Thus, we introduce as an objective function the mean α-weighted turnover time MTT_α of a solution S:

$$MTT_\alpha(S) = \frac{1}{n} \sum_{i=1}^{n} \alpha_i \, tt(por_i) \tag{1}$$

Given an instance of a MEX-MDP, an *optimal solution* with respect to a weight vector α is a solution S which *minimises* the objective function $MTT_\alpha(S)$. Two vectors α have been found to be interesting, namely data priority and data volume generated by the observations (see Cesta *et al.*, 2002a for a detailed description). The experimental evaluation in this paper considers the *Mean Turnover Time* (MTT) with $\alpha_i = 1, i = 1, \ldots, n$.

2.3 Complexity Analysis

This section analyses the complexity of MEX-MDP and in particular shows how the minimisation of the *Mean Turnover Time* of a problem instance is *NP-hard*. This is done by showing that a particular *NP-hard single-machine* scheduling problem can be reduced to MEX-MDP.

We call LP the scheduling problem $1|chains; r_i; pmtn|\sum C_i$ [1]. LP is a single-machine scheduling problem that is *NP-hard* (Lenstra, 2002) [2]. An instance of LP is composed of n activities, each one having a ready time r_i and a duration dur_i. In this problem *preemption* (*pmtn*) is allowed, hence the execution of an activity a_i can be suspended while a new one is resumed or started from scratch. Finally, the whole set of activities $\{a_i\}$ is partially ordered using the operator $\prec$, which relates the execution of a pair of activities. If $a_i \prec a_j$ then a_j can start only if activity a_i has been completed. Such a partial order

[1] According to the $\alpha|\beta|\gamma$-notation (Graham *et al.*, 1979).

[2] Referenced in http://www.mathematik.uni-osnabrueck.de/research/OR/class/

subdivides the set of activities into a set of separate *chains*, where each chain is a totally ordered subset of activities.

Theorem 1 *The* MEX-MDP *optimisation problem is NP-hard.*

Proof: The theorem is proved in two steps. Starting from LP, a new scheduling optimisation problem, named LP', is first defined and proved to be *NP-hard*. Second, LP' is reduced to MEX-MDP to prove its *NP-hardness*.

LP' differs from LP for how the ready times r'_i are defined. In particular, whereas in LP no constraints are imposed on the ready times r_i, in LP' the ready times along the activities of each chain are non-decreasing. Being part of a *chain* the execution of a generic activity a_i cannot start before the minimal instant of time t'_i. That is,

$$t'_i = \max \left\{ r_i, \max_{a_j \prec a_i} \left\{ r_j + dur_j + \sum_{a_j \prec a_k \prec a_i} dur_k \right\} \right\} \tag{2}$$

This means that in LP' the constraints $r'_i = t'_i$ are imposed for each activity a_i. Given a generic instance p of LP the corresponding instance p' of LP' is such that its optimal solution $\sum C'_i$ is also an optimal solution for p. Hence, by contradiction, if LP' is not *NP-hard*, then LP cannot be *NP-hard*. This proves that LP' is *NP-hard*.

As a second step we now reduce LP' to MEX-MDP. Let us consider the following relaxations on MEX-MDP: (i) each packet store has an infinite capacity; (ii) the communication channel has a uniform data rate dr over the total problem horizon; (iii) the problem horizon is not bounded; (iv) each por_i stores data in a single packet store and there is no pair of PORs that store data in the same packet store at the same time. We now reduced LP' to MEX-MDP. The single-machine corresponds to the communication channel. The number of packet stores equals the number of chains. Each chain of activities corresponds to a set of activities in MEX-MDP which stores data in the same packet store and has the same temporal total order imposed by the chains (remember that each packet store is a FIFO buffer). In particular, each activity a_i, with duration dur_i and ready time r'_i, holding the constraint (2), corresponds to a por_i which contains a single store activity with end time $e(por_i) = r'_i$, duration dur_i equal to the ratio between the data size of the por_i and the channel data rate dr. Finally, we observe that the completion time C'_i and the turnover time tt_i are equivalent objective functions. In fact, $C'_i = tt_i + r'_i$, hence a solution is optimal with respect to $\sum C'_i$ if and only if is optimal with respect to $\sum tt_i$. □

The complexity of MEX-MDP is a clear limitation to problem scalability in finding an optimal (or *near-optimal*) solution. For this reason, this paper proposes a set of heuristic strategies for solving MEX-MDP.

CSPsolver:
Input: A CSP instance
Output: A consistent assignment to all variables X_i

```
1. {
2.    while not(solved or infeasible) do
3.    {
4.       RemoveInconsistentValues
5.       SelectDecisionVariable
6.       SelectValueForVariable
7.    }
8. }
```

Figure 2. A backtrack-free CSP search procedure.

3. A CONSTRAINT-BASED MODEL FOR MEX-MDP

The approach we have pursued for the resolution of MEX-MDP is based on its representation as a CSP (Tsang, 1993), a generic problem solving technique that consists in modelling the problem as a set of variables $X = \{X_1, X_2, \ldots, X_n\}$, each associated with a domain D_i of values, and a set of constraints $C = \{C_1, C_2, \ldots, C_q\}$ denoting the legal combinations of values for the variables s.t. $C_i \subseteq D_1 \times D_2 \times \cdots \times D_n$. A solution to the CSP consists of assigning to each variable one of its possible values so that all the constraints are satisfied. Figure 2 shows a generic, backtrack-free algorithm for solving a CSP instance. The resolution process can be seen as an iterative search procedure where the current (partial) solution is extended on each cycle by assigning a value to an unassigned variable.

As a new decision is made during the search, a set of automatic deductions, called "propagation rules", can remove elements from domains D_i which cannot be contained in any feasible extension of the current partial solution (Step 4 of the algorithm). In general it is not possible to remove all inconsistent values through propagation alone. Choices have to be made among possible values for some variables, giving rise to the need for *variables* and *values ordering* heuristics (Steps 5 and 6 in Figure 2).

A CSP representation for a problem should focus on its important features. In the case of MEX-MDP, the following characteristics have been selected:

1. the temporal horizon $\mathcal{H} = [0, H]$
2. the amount of data stored at the end time of each operation
3. the channel communication windows

4. the finite capacity c_i of each memory bank pks_i and its FIFO behaviour.

It is worth noting that the FIFO behaviour of the packet stores allows us to make an important simplification. In fact, it is possible to consider both the data in input and those in output to/from the memory as flows of data, neglecting the information about which operations those data refer to. In this way, given a generic time window over the communication channel, it is possible to split the problem into two levels of abstraction. A first one, where we just consider the constraints on the flows of data: for each generic dump window and each packet store the amount of residual data in the packet store should not exceed its capacity. And a second level, where a sequence of memory dump operations (generation of data packets) is generated over the communication link. In particular, we divide the temporal horizon $\mathcal{H}$ into a set of contiguous temporal windows $w_j = (t_{j-1}, t_j]$, with $j = 1 \ldots m$, according to the domain's *significant events*.

Significant events are assumed to be the start and the end of the temporal horizon, the time instants where a memory reservation on a packet store is performed, and the time instants where a change on the channel data rate is operated. This partition is constructed in order to have such events only at the edges of the windows; this allows us to consider temporal intervals, w_j, in which store operations do not happen (except for its upper bound t_j) and the data rate is constant.

Figure 3 sketches a representation of the problem domain, where the temporal horizon is partitioned into contiguous time windows. At data dump level, the set of CSP decision variables are defined according to this set of windows, such that, for each packet store pks_i ($i = 1, \ldots, n$) and for each time window w_j ($j = 1, \ldots, m$) the following variables are introduced:

$$\delta_{ij} \qquad i = 1, \ldots, n,\ j = 1, \ldots, m \tag{3}$$

each one with an initial domain $[0, \infty)$. They represent the amount of data dumped from the packet store pks_i within a window w_j. To formally represent the domain constraints, for each packet store pks_i ($i = 1, \ldots, n$) and for each time window w_j ($j = 1, \ldots, m$) some additional definitions are needed:

- d_{ij}: amount of data memorised in the packet store pks_i at t_j, where the variables $d_{i0} \leq c_i$ represent the initial level of data in the packet store pks_i;

- l_{ij}: maximal level (amount of data stored) allowed at the instant t_j for the packet store pks_i, $l_{ij} \in [0, c_i]$;

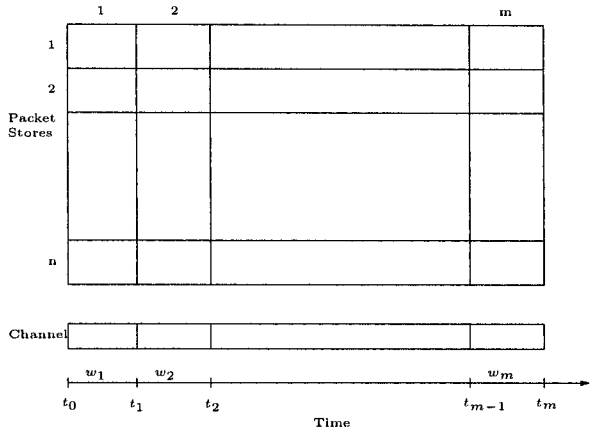


Figure 3. Packet stores and channel vs time windows.

- b_j: maximal dumping capacity available [3] in w_j.

In the following, the values d_{ij}, l_{ij} and b_j are used to integrate the CSP model of MEX-MDP with the definition of constraints the variables δ_{ij} should satisfy.

3.1 Constraints on the Decision Variables

Three classes of constraints are defined on the decision variables. A first constraint captures the fact that for each window w_j the difference between the amount of generated data and the amount of dumped data cannot exceed l_{ij} the maximal imposed level in the window (*overwriting*). Additionally, the dumped data cannot exceed the generated data (*overdumping*). We define the following inequalities (4) as *conservative constraints*:

$$\begin{aligned} \sum_{k=0}^{j} d_{ik} - \sum_{k=1}^{j} \delta_{ik} &\leq l_{ij} \\ \sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j} \delta_{ik} &\geq 0 \end{aligned} \tag{4}$$

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

[3] In the MARS EXPRESS domain a reasonable hypothesis is that the set of communication links are "mutually exclusive", hence a single link representation with variable data rate is enough to represent the real situation.

A second class of constraints considers the dumping capacity imposed by the communication channel. The following inequalities, called *downlink constraints*, state that for each window w_j it is not possible to dump more data than the available capacity b_j, $j = 1, \ldots, m$:

$$0 \leq \sum_{i=1}^{n} \delta_{ij} \leq b_j \tag{5}$$

Each decision variable δ_{ij} has a potential interval of feasible values $[lb_{\delta_{ij}}, ub_{\delta_{ij}}]$ defined by its lower and upper bounds $lb_{\delta_{ij}}$ and $ub_{\delta_{ij}}$. For $i = 1, \ldots, n$, $j = 1, \ldots, m$:

$$lb_{\delta_{ij}} = \max\left\{0, \max\left\{0, \sum_{k=0}^{j} d_{ik} - l_{ij}\right\} - \max\left\{\sum_{k=1}^{j-1} b_k, \sum_{k=0}^{j-2} d_{ik}\right\}\right\} \tag{6}$$

$$ub_{\delta_{ij}} = \min\left\{b_j, \sum_{k=0}^{j-1} d_{ik} - \max\left\{0, \sum_{k=0}^{j-1} d_{ik} - l_{i(j-1)}\right\}\right\} \tag{7}$$

Equation (6) states that a lower bound of δ_{ij} is represented by the difference between the amount of data generated at t_j over the packet store capacity l_{ij} ($\sum_{k=0}^{j} d_{ik} - l_{ij}$) and the maximal amount of data which is "downloadable" by t_{j-1} ($\max\{\sum_{k=1}^{j-1} b_k, \sum_{k=0}^{j-2} d_{i,k}\}$). Equation (7) claims that an upper bound of δ_{ij} is the minimal value between b_j and the maximal amount of data which is really "downloadable" within the window w_j ($\sum_{k=0}^{j-1} d_{ik} - \max\{0, \sum_{k=0}^{j-1} d_{ik} - l_{i(j-1)}\}$). Grounded on these constraints, a set of propagation rules (or domain filtering rules) are defined to further reduce the domain intervals $[lb_{\delta_{ij}}, ub_{\delta_{ij}}]$.

3.2 Domain Filtering Rules

In a constraint-based solver, domain filtering rules have a fundamental role to prune the search space by removing inconsistent values in the decision variable domains. The next two theorems allow us to generate propagation rules from the conservative and the downlink constraints, obtaining a reduction of the domain interval $[lb_{\delta_{ij}}, ub_{\delta_{ij}}]$ of the involved variables δ_{ij}.

Theorem 2 *For each decision variable δ_{ik}, the set of its feasible values is contained in the interval*

$$\delta_{ik} \in \left[\sum_{j=1}^{p} d_{ij} - l_{ip} - \sum_{j=1, j\neq k}^{p} ub_{\delta_{ij}}, \sum_{j=1}^{p} d_{ij} - \sum_{j=1, j\neq k}^{p} lb_{\delta_{ij}}\right]$$

for $i = 1, \ldots, n$, $p = 1, \ldots, m$, and $k = 1, \ldots, p$.

Proof: By contradiction, let δ'_{ik} be a value such that $\delta'_{ik} > \sum_{j=1}^{p} d_{ij} - \sum_{j=1,j\neq k}^{p} lb_{\delta_{ij}}$; this implies to dump from the packet store pks_i an amount of data $\delta'_{ik} + \sum_{j=1,j\neq k}^{p} lb_{\delta_{ij}} > \sum_{j=1}^{p} d_{ij}$ greater than the amount of data stored, $\sum_{j=1}^{p} d_{ij}$. On the other hand, taking a value $\delta''_{ik} < \sum_{j=1}^{p} d_{ij} - l_{ip} - \sum_{j=1,j\neq k}^{p} ub_{\delta_{ij}}$ means to store an amount of data

$$\sum_{j=1}^{p} d_{ij} - \delta''_{ik} - \sum_{j=1,j\neq k}^{p} ub_{\delta_{ij}} > l_{ip}$$

greater than the maximal level allowed. □

Theorem 3 *The feasible values for each decision variable δ_{kj} are bounded by the following upper bound:*

$$\delta_{kj} \leq b_j - \sum_{i=1,i\neq k}^{n} lb_{\delta_{ij}}$$

for $k = 1, \ldots, n$, and $j = 1, \ldots, m$.

Proof: Let $\delta_{k'j}$ be a value such that $\delta_{k'j} > b_j - \sum_{i=1,i\neq k'}^{n} lb_{\delta_{ij}}$; it is possible to verify that the downlink constraint (5)

$$\sum_{i=1}^{n} lb_{\delta_{ij}} = \delta_{k'j} + \sum_{j=1,j\neq k'}^{n} lb_{\delta_{ij}} > b_j$$

is violated. □

From Theorem 2 the following two propagation rules can be defined:

$$ub_{\delta_{ik}} = \min \left\{ ub_{\delta_{ik}}, \sum_{j=1}^{p} d_{ij} - \sum_{j=1,j\neq k}^{p} lb_{\delta_{ij}} \right\} \tag{8}$$

$$lb_{\delta_{ik}} = \max \left\{ lb_{\delta_{ik}}, \sum_{j=1}^{p} d_{ij} - l_{ip} - \sum_{j=1,j\neq k}^{p} ub_{\delta_{ij}} \right\} \tag{9}$$

for $i = 1, \ldots, n$, $p = 1, \ldots, m$, and $k = 1, \ldots, p$. A further rule stems from Theorem 3:

$$ub_{\delta_{ik}} = \min \left\{ ub_{\delta_{ik}}, b_j - \sum_{i=1,i\neq k}^{n} lb_{\delta_{ij}} \right\} \tag{10}$$

for $i = 1, \ldots, n$ and $k = 1, \ldots, m$.

The application of these filtering rules updates the domain associated with the variable δ_{ij}, both increasing the lower bound $lb_{\delta_{ij}}$ and reducing the upper bound $ub_{\delta_{ij}}$. In the case that the condition $lb_{\delta_{ij}} \leq ub_{\delta_{ij}}$ is violated for at least one window w_j, the current situation will not admit a feasible solution.

It is worth remarking that, due to their incompleteness, the application of the rules (8)–(10) does not guarantee that any assignment of $\delta_{ij} \in [lb_{\delta_{ij}}, ub_{\delta_{ij}}]$ is a solution of the MEX-MDP instance. In Section 5 we will show how these rules affect our solving methods and highlight their role in achieving a feasible solution.

4. SOLVING METHODS

The complexity of the MEX-MDP is a strong limitation to problem scalability in finding optimal (or *near-optimal*) solutions under tight temporal bounds on the computation time. For this reason in this section we propose two different solving heuristic strategies: (a) a *core* greedy procedure that uses the propagation rules defined in Section 3 and (b) an iterative random sampling strategy based on such greedy procedure. As a matter of fact, in many cases a randomised search algorithm is faster and simpler than its deterministic and systematic counterpart. In Section 5 we will show that a few iterations of the random sampling algorithm are able to find high quality solutions quite close to a solution lower bound.

4.1 The Greedy Solver

The core procedure for solving MEX-MDP is shown in Figure 4. The algorithm, called *GreedyCspSolver()*, takes as input a MEX-MDP instance (that is, a temporal horizon H, a set of store activities, a description of the communication links, and the set of packet stores) and returns as output either a sequence S of memory dumps over the communication link consistent with all the domain constraints or a failure in case the procedure could not find a feasible solution. This algorithm is based on the above CSP model and uses the two levels of abstraction for the MEX-MDP problem previously described, as shown in the two-step procedure described in what follows.

Data dump level. Figure 5 shows the algorithm *MakeConsistentDataDump()*. At first (Step 2) a propagation procedure is called. This procedure implements the propagation rules described above (see Section 3) and basically sets the domains $[lb_{\delta_{ij}}, ub_{\delta_{ij}}]$ of possible values for all the decision variables δ_{ij}. In addition, each time the algorithm takes a decision (Step 5), the *Propagation()* procedure is called again in order to further reduce the domains $[lb_{\delta_{ij}}, ub_{\delta_{ij}}]$ (Step 6).

GreedyCspSolver:

Input: $mexmdp$ instance

Output: sequence S of memory dumps over the communication links

```
1. {
2.    MakeConsistentDataDump(mexmdp)
3.    GenerateDataPackets(mexmdp)
4. }
```

Figure 4. The greedy constraint-based solver.

MakeConsistentDataDump:

Input: $mexmdp$ instance, random $seed$

Output: a consistent assignment of the set of decision variables δ_{ij}

```
1. {
2.    Propagation(mexmdp)
3.    j ← 1
4.    while (j ≤ m and Feasible(mexmdp)) {
5.       AssignDecisionVariablesΔ(j)
6.       Propagation(mexmdp)
7.       j=j+1
8.    }
9. }
```

Figure 5. The constraint-based algorithm to assign the δ_{ij} variables.

If the condition $lb_{\delta_{ij}} > ub_{\delta_{ij}}$ holds for at least one variable δ_{ij}, the algorithm terminates without producing a solution. The function *Feasible()* just reports the feasibility state of the current partial solution under construction. Basically, the algorithm considers the set of windows $w_1, w_2, \ldots, w_m$ in increasing order of time and, for each window w_j, it sets the amount of data b_j that can be dumped within the window using *AssignDecisionVariables*$\Delta()$, the sub-procedure that iteratively executes the following steps:

1. Select a packet store according to a given priority rule. In the current algorithm, two rules are implemented: a first one selects the packet store with the highest percentage of data volume (CFF, Closest to Fill First); a second selects the packet store with the highest priority (HPF, Highest Priority First); in the case of same priority, the packet store with the smallest store as outcome data is chosen.

2. Assign an amount of data to be dumped from the selected packets store. Such an amount of data is computed according to the upper bound of δ_{ij}, $ub_{\delta_{ij}}$, and the remaining availability of the channel in the window w_j of b_j:

$$\delta_{ij} = \min \left\{ ub_{\delta_{ij}}, b_j - \sum_{l \neq i} \delta_{lj} \right\}$$

3. Update the lower bound of the domain of the involved decision variable and the availability of the channel b_j. This is accomplished using the rules described in Section 3.2.

These steps are executed, for each window w_j, until a solution is achieved or a failure state occurs.

Packet level. The second step inside *GreedyCspSolver()* is called *GenerateDataPackets()* and creates the final solution S, that is the sequence of memory dump operations. It is worth remarking that, given a solution at the Data Dump level, finding a final solution S can be accomplished by a polynomial algorithm. In fact, when all the variables δ_{ij} are consistently assigned, it is possible to generate the sequence of data dumping operations without the risk of finding inconsistent states. The procedure works in an analogous way to *MakeConsistentDataDump()* with the difference that no propagation function is called. In particular, the algorithm considers the set of windows $w_1, w_2, \ldots, w_m$ in increasing order of time, such that for each packet store pks_i and window w_j, given the value of the decision variable δ_{ij}, it generates a sequence of memory dumping operations (stream of data packets) for an amount that equals that value.

Similarly to the previous level, it is also possible to define different priority rules. To obtain the results is Section 5 we have implemented the following heuristic: select the packet store with the smallest value of the decision variable δ_{ij} first (SDF—Smallest to Dump First).

4.2 Iterative Random Sampling

In this section we describe the meta-heuristic strategy for optimisation using the greedy procedure just discussed. In short, this method iteratively performs a random sampling of the space of feasible solutions until a *termination condition* is met. A solution is sampled by a randomised greedy procedure which incrementally generates a solution or a failure. The overall process generates a stream of feasible solutions with different values of the objective function together with a set of failures. When the procedure stops, the best solution found is returned.

IterativeRandomSampling:

Input: $mexmdp$ instance, termination conditions

Output: best solution found S^*

```
1. {
2.    S* ← ∅
3.    while (termination conditions not met) {
4.       S ← GreedyRandomSampling(mexmdp)
5.       UpdateBestSolution(S, S*)
6.    }
7. }
```

Figure 6. The iterative random sampling algorithm.

It is worth noting that the greedy solver is based on the composition of a dispatching strategy that takes decisions proceeding ahead in time and the effects of the propagation rules on the CSP representation that propagate decision forward and backward on the structure, adjusting values according to problem constraints. The integration of a propagation mechanism inside the random sampling procedure has the main benefit of increasing the probability of finding feasible solutions during each iteration. This property is very useful when the optimisation problem is formulated within a set of tight constraints, such that the set of feasible solutions is *sparse* over the search space.

The complete algorithm is given in Figure 6. It takes as input an instance of the MEX-MDP problem and a set of termination conditions (for example, a maximal number of iterations or a CPU time bound). The output of the algorithm is either the best solution found S^* (sequence of memory dumps over the communication link) consistent with all the domain constraints or a failure, in the case no iteration finds a feasible solution.

The sampling procedure is composed of two steps: first, a procedure called *GreedyRandomSampling()* builds a solution S, while a second step updates the best solution S^* found during the search. *GreedyRandomSampling()* (see Figure 7) is very similar to *GreedyCspSolver()* shown in Figure 4, the only difference being the priority rule used inside the step *MakeConsistentDataDump()*: in this case the rule is randomly generated. In particular, for each window w_j, a set of random priority values are assigned to each packet store, then the packet stores are selected with the same HPF rule (the packet store with the highest priority first) defined in Section 4.1.

GreedyRandomSampling:

Input: $mexmdp$ instance

Output: sequence S of memory dumps over the communication links

```
1. {
2.    seed ← InitialiseRandomSeed()
3.    MakeConsistentDataDump(mexmdp, seed)
4.    GenerateDataPackets(mexmdp)
5. }
```

Figure 7. The greedy random sampling procedure.

5. EXPERIMENTAL EVALUATION

In the first part of the paper we have introduced a combination of solving techniques: (a) a CSP formalisation and the associated set of propagation rules, (b) a greedy solver and (c) an iterative sampling strategy based on a randomisation of the greedy solver. In this section we present an experimental analysis to show the effectiveness of the approach. Before presenting the evaluation itself we discuss a problem analysis that has been followed to set up the experimental setting.

Finding meaningful benchmark sets has been an additional problem because in the period of our work for ESA (December 2000–May 2002) no real data were available. For this reason a problem generator has been built (described in Cesta *et al.*, 2002a) that creates problem instances from a set of test data provided by ESA mission planning experts. The generator uses these data in order to create random problems on different configurations of the spacecraft domain. Using the problem generator we created 27 MEX-MDP problem instances with the following domain parameters: 1 spacecraft housekeeping packet store, 1 science housekeeping packet store, 11 science packet stores, 8 payloads and a channel data rate at 228 Kbps. From the set of the 27 problems a subset of 6 MEX-MDP instances has been selected with a number of observation requests ranging from 12 to 96 (corresponding to three days of satellite work). The remaining ones are not considered because either the number of payload operation requests was too small or there was too little interaction among the packet stores. From this *core* benchmark set, we generated four new benchmark sets by removing/adding payload operations from the original six problems, reducing the packet stores capacities and/or the transmission data rates [4].

[4] The benchmark sets, named B1–B4, are available at http://mexar.istc.cnr.it/

To analyse the generated benchmarks we have developed a broad classification schema of the MEX-MDP problems, that allows us to distinguish between *easy* and *hard* instances. In addition, to better understand the effectiveness of the solvers we have identified a lower bound for the optimal value of the mean turnover time. This allows us to evaluate the *distance* between an optimal solution and the corresponding best solution found by any of the proposed solving methods.

5.1 Analysing Problem Features

To analyse the MEX-MDP instances and understand the factors that generate *hard* instances we have defined two parameters: the *Communication Coefficient (CC)* and the *Load Coefficient (LC)*.

Given a MEX-MDP instance, the *CC* is the ratio between the total volume of data generated, both by the set of payload operations and by the housekeeping activities, and the total data volume downloadable in the time windows $[\min_i\{s(por_i)\}, \max_i\{e(por_i)\}]$, where, $\min_i\{s(por_i)\}$ ($\max_i\{e(por_i)\}$) is the minimal start time (maximal end time) among the set of store activities in the payload operations set.

The second parameter, the *LC*, compares for each packet store pks_j the total amount of stored data Q_j with the packet store capacity c_j. We define the load coefficients $LC_{\max}$ as

$$LC_{\max} = \max_j\{Q_j/c_j\} \tag{11}$$

Using CC and $LC_{\max}$ a broad classification of problem instances in four different classes is defined that allows a qualitative mapping in terms of difficulty (see Figure 8):

1. $CC \leq 1$ and $LC_{\max} \leq 1$. These are the *easiest* instances because there is surely time to dump all the data ($CC \leq 1$) and also it is not possible to *overwrite* data in the packet stores ($LC_{\max} \leq 1$).

2. $CC \leq 1$ and $LC_{\max} > 1$. In this case the different instances of MEX-MDP could present a risk of overwriting ($LC_{\max} > 1$).

3. $CC > 1$ and $LC_{\max} \leq 1$. This class of problems show another kind of criticality, there is no risk of overwriting, however the higher is the value CC, the higher is the probability that a subset of packet stores remains with some residual data.

4. $CC > 1$ and $LC_{\max} > 1$. This class of instances exhibits both kinds of criticality. We expect that the most difficult instances are in this subset.

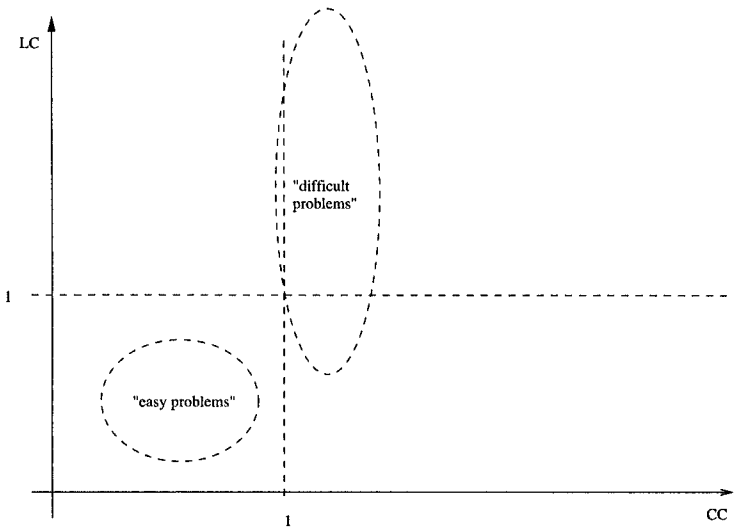


Figure 8. Problem classification.

5.2 A Lower Bound for the Objective Function

When working on the hard instances of MEX-MDP it is not easy to obtain a reference solution to be compared with the heuristic solution produced with our algorithms. For this reason, for the minimisation problem we compare the current best value of the objective function with a *lower bound* [5].

In general, computing a lower bound involves two separate steps: first, a relaxation of the problem is defined (some of the original problem constraints are removed); second, the optimal value of the objective function is computed (in this case the value MTT). Since the number of solutions of the *relaxed* problem is greater or equal to the number of solutions of the original problem, the optimal solution of the *relaxed* problem has an objective function value lower than or equal to the optimal solution of the original problem. To find a representative lower bound we consider two relaxed formulations of MEX-MDP:

- First, we consider the relaxed version of the problem such that each packet store has an infinite capacity and there are no chain constraints among the activities. Under this hypothesis, the problem can be reduced to the classical optimisation scheduling problem of minimising the mean

[5] In contrast, in the case of a maximisation problem an upper bound is requested.

flow time (in our case called MTT) on a *single machine* (i.e. the communication channel) where *preemption* is allowed. For this problem a polynomial strategy which gives the optimal value is given by the Shortest Remaining Processing Time (SRPT) (Lenstra *et al.*, 1977).

- The second relaxed version of MEX-MDP considers that each packet store has a dedicated communication channel with identical characteristics to the original one. In this way any linear sequence of activities (chain) which stores data in a packet store has a dedicated channel for data dumping and the optimal solution can again be computed in polynomial time.

Based on these two relaxed versions of MEX-MDP, a lower bound has been defined as the maximum mean turnover time between the two relaxed formulations of the problem.

5.3 Experimental Results

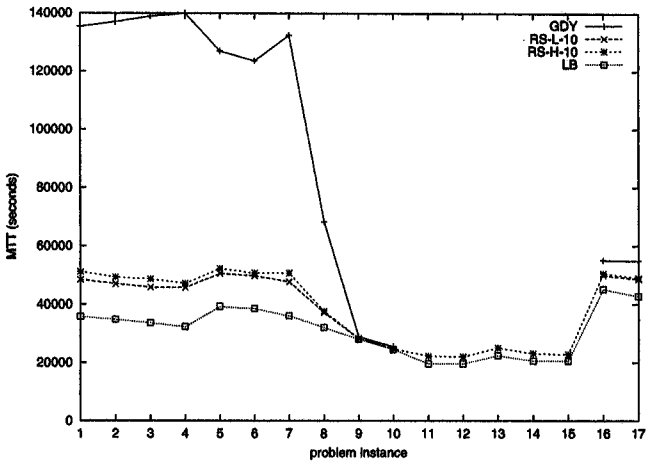
In this section we present the results for two of the benchmark sets, B4 and B1, each composed of 17 instances. With respect to the qualitative classification just introduced, B4 instances are mapped *around* the "difficult problems" ($CC > 1$ and $LC_{\max} > 1$), whereas B1 instances are *around* the "easy problem" zone. Somehow the two benchmark sets explore different facets of the problem. Using them we can both clarify the role of the propagation rules and analyse the effectiveness of the iterative sampling procedure in finding near optimal solutions.

All the algorithms presented in this paper are implemented in Java and the results presented in the tables and in the figures are obtained on a PC Athlon 1800 MHz machine under Windows XP. We represent three types of results:

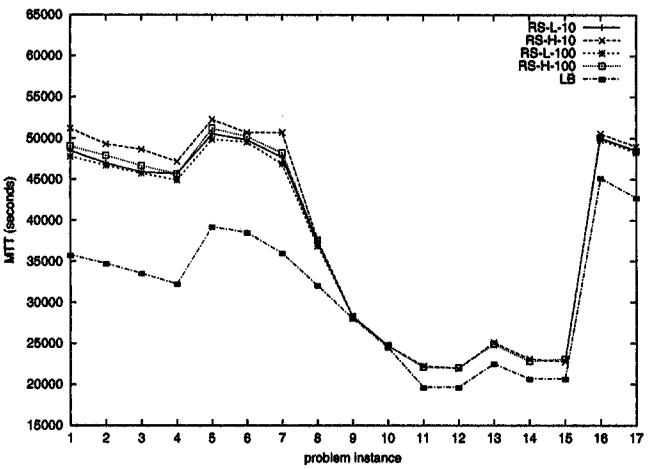
- the lower bounds of the MTT values;
- the values generated by the greedy (one-pass) strategy;
- the best MTT values obtained with the iterative sampling optimisation strategy.

It is worth noting that inside the random sampling procedure it is possible to use the different propagation rules defined in Section 3 with noticeable differences in performance both with respect to the CPU time and to the number of solved problems.

We consider two versions of the iterative random sampling algorithm: (a) one, identified by letter "H", for "Heavy", which uses the set of propagation rules (8)–(10) of Section 3.2; (b) another, identified by letter "L", for "Light", that uses only the rule (10). It is worth recalling that the L and H versions require



(a) MTT values with a bound of 10 seconds.



(b) MTT values with bounds of 10 and 100 seconds.

Figure 9. Performance on benchmark B4.

different computational complexity. In fact, rule (10) takes only $\mathrm{O}(n)$, where n is the number of packet stores. This rule has only a local effect within a single transmission window w_j. On the other hand, the other two rules have a computational complexity $\mathrm{O}(m^2)$—with m number of windows on the horizon—because they propagate the effects of a single choice over the entire horizon.

Experiment 1. Figure 9 shows the algorithms' performance on benchmark B4 when a time limit of 10 seconds is imposed on the computational time. In particular, problem instances are labelled with integer numbers (1–17) and the following convention is adopted for the solving methods: GDY labels the results obtained by the greedy one-pass procedure, RS-L-10 the results generated with the randomised strategy which uses only the propagation rule (10), RS-H-10 uses all the propagation rules, and LB labels the lower bound values.

We start by focusing our attention on the effects of the propagation rules (8)–(10) on the iterative sampling algorithm. It is worth highlighting that within the same temporal bound the two versions of the iterative sampling algorithm perform a different number of iterations. In fact, for RS-L-10, which uses a "lighter" domain filtering rule, it is possible to make more iterations than RS-H-10 within the same period of time [6].

From the results in Figure 9(a) we also observe a clear partition between the first eight instances and the remaining ones. For the former there is the *possibility* to find better quality solutions with respect to the greedy performance. In fact, in Figure 9(a) both the two random iterative sampling strategies improve up to 70% over the greedy strategy and obtain results quite close to the lower bounds.

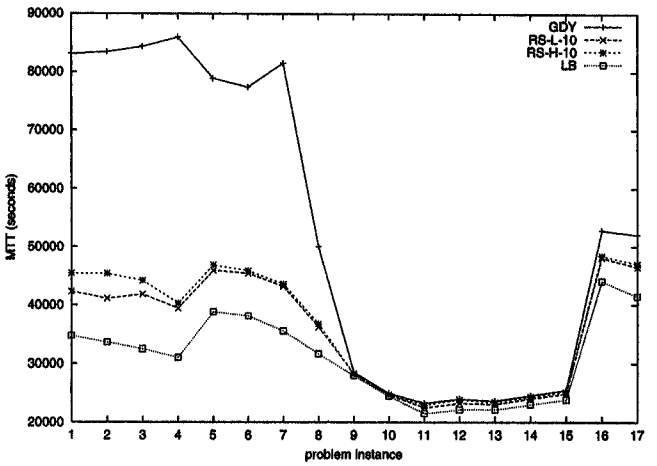
A further interesting aspect is that even if RS-L-10 shows an apparently better behaviour than RS-H-10 it is not able to find a feasible solution for all the instances. In fact, without the use of the propagation rules (8) and (9), the problem instances 11–15 remain unsolved (in this case we do not report any results on the graph of Figure 9(a)). This confirms the usefulness of the propagation rules (8) and (9), when the problems have tight constraints, as in the case of the problems B4.

Experiment 2. In order to better understand the trade-off between the use of the propagation rules and the number of iterations, Figure 9(b) shows an additional set of results, without the greedy algorithm and with the addition of the iterative sampling strategy where a 100 seconds time limit has been used [7].

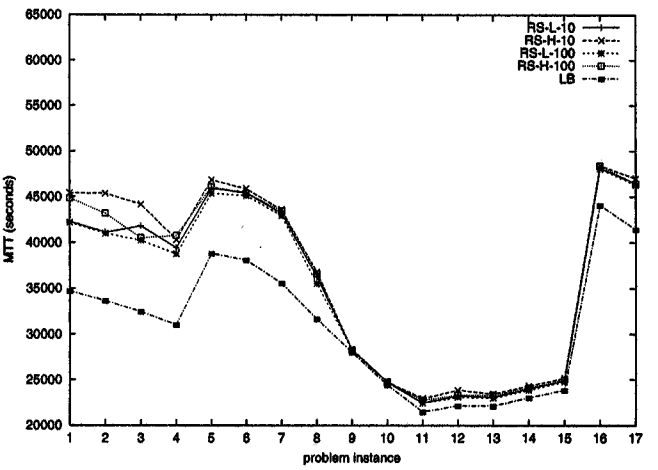
As it is possible to see, the same set of problems remain unsolved (problems 11–15), when propagation rules (8) and (9) are not used. Moreover, we observe that, also in this case, within the subset of problems solved by all the strategies, the best results are provided by the iterative sampling algorithm without propagation rules, RS-L-100. In other words, when a problem has not tight constraints (all the algorithms solve it), within an iterative sampling strategy,

[6] Each run of the core greedy procedure takes in average 0.21 seconds with the use of propagation rules and 0.01 seconds without.

[7] The same data are also shown in Table 1, where the character "-" indicates unsolved instances. For each instance the number of scientific observations is also given (column *size*).



(a) MTT values with a bound of 10 seconds.



(b) Best MTT values rules with bounds of 10 and 100 seconds.

Figure 10. Performance on benchmark B1.

the best choice is a light core greedy strategy. This result is confirmed by the experiments carried out on benchmark B1.

Experiment 3. In Figure 10 we show the performance on the benchmark B1 by using the same set of algorithms used for B4. A similar pattern of results can be observed, the only difference being that the whole set of problems is

Table 1. MTT values with bounds of 10 and 100 seconds w.r.t. benchmark B4.

#	Size	RS-L-10	RS-H-10	RS-L-100	RS-H-100	LB
1	60	48476	51160	47774	49020	35775
2	66	46971	49320	46678	47886	34741
3	86	45878	48636	45715	46660	33544
4	76	45682	47150	44886	45630	32239
5	87	50496	52251	49842	51194	39220
6	66	49804	50681	49519	50185	38529
7	66	47667	50697	46851	48217	36001
8	58	37220	37666	36839	37509	32064
9	81	28221	28296	28221	28221	28037
10	62	24686	24686	24686	24686	24476
11	66	-	22216	-	22045	19615
12	91	-	21948	-	21948	19615
13	96	-	25079	-	24863	22480
14	56	-	23083	-	22780	20641
15	71	-	22760	-	22760	20641
16	15	50021	50537	49818	50100	45167
17	12	48538	48970	48294	48452	42738

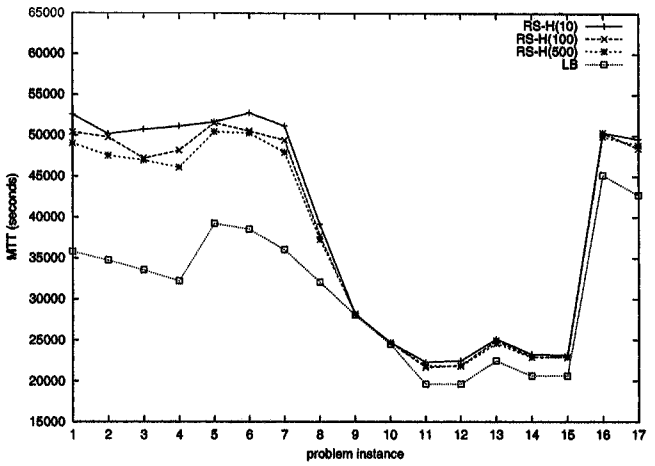
solved by all the strategies. In fact since B1 belongs to the "easy problems" category (see Figure 8) the iterative algorithm which uses the rule (10) ensures that we obtain the best solution over the set of all instances.

Experiment 4. To conclude, a complementary perspective of the experimental results has been obtained by evaluating the performance of the iterative sampling algorithms as a function of the number of iterations. In other words, we evaluate the *speed* of the iterative sampling strategies to converge toward optimal solutions.

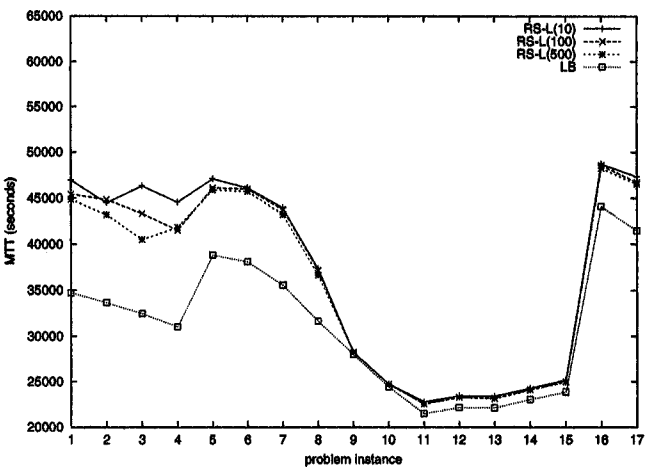
Figure 11(a) shows, for the benchmark B4, the performance of the iterative sampling algorithm RS-H(n_{iter}), where n_{iter} represents the number of iterations.

On benchmark B1 the iterative algorithm which uses only rule (10), RS-L(n_{iter}), is used (Figure 11(b)). The results are given for three different number of iterations: 10, 100 and 500.

We observe that even if the best results are generated with a limit of 500 iterations, the 10 iterations ensure high quality solutions. This confirms that the proposed randomised search algorithm is an efficient method for finding high-quality solutions.



(a) MTT values for benchmark B4.



(b) MTT values for benchmark B1.

Figure 11. Efficiency of the iterative sampling algorithms.

6. DISCUSSION

The goal of the study we have conducted for ESA has been not only to analyse the problem, but also to create an interactive software tool for solving MEX-MDPS. Such a tool, named MEXAR, is described in Cesta *et al.* (2002a, 2003). The system has a bipartite architecture: (a) a problem solving module

contains a set of CSP-based algorithms and (b) an interaction module facilitates the interaction of human planners with different aspects of a problem and its solutions. The choice of following a CSP approach has been driven not only by our in-house expertise but also by the need to interface the solver with the user interaction module. In fact both parts of the software work on the internal symbolic model developed in terms of *resources* and *activities* (Section 2).

It is worth noting that MEX-MDP is a quite difficult scheduling problem, because it involves preemption as well as resource constraints and a complex set of temporal constraints, such as ready times, deadlines and chain constraints. Even if some works in the literature (Verfaillie and Lemaitre, 2001; Bensana *et al.*, 1999) describe problem domains which may resemble the MEX-MDP domain, none of them matches exactly its features. The problem can be faced by adapting existing optimisation techniques such as Integer Linear Programming, Constraint Programming, and many others. The integration of different techniques is the approach we have followed. In this paper we have described the combination of constraint satisfaction and iterative random sampling procedure while in a companion work (Oddi *et al.*, 2003) we have additionally proposed a tabu search procedure to improve the mean turnover time of a MEX-MDP. In a more recent work (Oddi and Policella, 2004) a different characteristic of a solution is considered, the so-called *robustness*. This measures the ability of a solution to *adsorb* unexpected modifications to the problem specification and to maintain consistency. In this work the main goal is a balanced distribution of data on the various packed stores to minimise peaks in memory usage. As expected, this different optimisation criterion worsens the mean turnover time measure.

There are several additional directions that have not been explored. Among them some are on our agenda for the immediate future:

(a) Improvements in the integration of constraint programming techniques. For example, some existing results in single-machine scheduling with preemption (e.g., those in Baptiste *et al.*, 2001) might be used in conjunction with results on consumable resource constraint propagation (Laborie, 2003), since packet stores can be seen as a particular kind of such resources.

(b) Definition of new GRASP-like strategies (Feo and Resende, 1995; Resende and Ribeiro, 2002). The idea is to extend the iterative sampling procedure presented here with a local search step to be executed each time the greedy step has sampled a solution. Somehow the goal is to improve the directions in Oddi *et al.* (2003) relying on the simple and powerful idea that the greedy step samples different solutions in distant areas of the search space while the local search operates a refinement in order to further improve the objective function.

(c) The definition of multi-objective optimisation algorithms in line, for example, with Dasgupta *et al.* (1999). This can be used to develop algorithms able to take into account various optimisation criteria, including the definition of robustness, in an integrated way.

7. CONCLUSIONS

This paper has introduced a challenging scheduling problem arising in a spacecraft domain called the MARS EXPRESS Memory Dumping Problem (MEX-MDP). Even though the problem comes from a specific study, its characteristics are quite general and many of the conclusions reported in this paper can be extended to other space programs. In this respect, it is worth noting that two of the next ESA missions, namely ROSETTA and VENUS EXPRESS, will adopt spacecrafts that use a model of on-board memory similar to the one of MARS EXPRESS.

The paper contains several contributions:

(a) it has defined a new scheduling problem coming from a real domain and analysed its computational complexity;

(b) it has proposed a model of the problem as a CSP and two different solving heuristic strategies: a core greedy procedure based on the propagation rules defined in Section 3 and an iterative random sampling optimisation strategy which uses the basic core greedy procedure;

(c) it has built an experimental setting for the problem based on realistic problem descriptions and shown the effectiveness of the solving algorithms according to different experimental perspectives.

ACKNOWLEDGMENTS

This work has been supported by the European Space Agency (ESA-ESOC) under contract No. 14709/00/D/IM. The authors would like to thank the ESA-ESOC personnel that have been involved in the MEXAR study. In particular, they are grateful to the project officer Fabienne Delhaise and to the MARS-EXPRESS mission planners Michel Denis, Pattam Jayaraman, Alan Moorhouse and Erhard Rabenau for introducing them to the MARS-EXPRESS world and for a number of prompt clarifications during the difficult steps of the study. They also would like to thank Professor J. K. Lenstra for his useful notes about the complexity of the single-machine scheduling problem.

References

Baptiste, P., Le Pape, C. and Nuijten, W. (2001) *Constraint-Based Scheduling*. Kluwer, Dordrecht.

Bensana, E., Lemaitre, M. and Verfaillie, G. (1999) Earth observation satellite management. *Constraints: An International Journal*, **4**:293–299.

Cesta, A., Cortellessa, G., Oddi, A. and Policella, N. (2003) A CSP-based interactive decision aid for space mission planning. In *Proceedings of AI*IA-03,* Lecture Notes in Artificial Intelligence, Vol. 2829, Springer, Berlin.

Cesta, A., Oddi, A., Cortellessa, G. and Policella, N. (2002a) Automating the generation of spacecraft downlink operations in MARS EXPRESS: Analysis, algorithms and an interactive solution aid. *Technical Report* MEXAR-TR-02-10 (Project Final Report), ISTC-CNR [PST], Italian National Research Council.

Cesta, A., Oddi, A. and Smith, S. F. (2002b) A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, **8**:109–135.

Dasgupta, D., Chakrabarti, P. and DeSarkar, S. C. (1999) *Multiobjective Heuristic Search.* Vieweg, Braunschweig.

Feo, T. A. and Resende, M. (1995) Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, **4**:287–326.

Laborie, P. (2003) Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, **2**:151–188.

Lenstra, J. K. (2002) Notes from Berkeley, July 11, 1980. Personal Communication.

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

Motwani, S. and Raghavan, P. (1995) *Randomized Algorithms.* Cambrige University Press, New York.

Nuijten, W. and Aarts, E. (1996) A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, **90**:269–284.

Oddi, A. and Policella, N. (2004) A max-flow approach for improving robustness in a spacecraft downlink schedule. In *Proceedings of the 4th International Workshop on Planning and Scheduling for Space, IWPSS'04*, ESA-ESOC, Darmstadt, Germany.

Oddi, A., Policella, N., Cesta, A. and Cortellessa, G. (2003) Generating high quality schedules for a spacecraft memory downlink problem. In *Principles and Practice of Constraint Programming, 9th International Conference, CP 2003*, Lecture Notes in Computer Science, Vol. 2833, F. Rossi (Ed.), Springer, Berlin, pp. 570–584.

Oddi, A. and Smith, S. (1997) Stochastic procedures for generating feasible schedules. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI-97.* AAAI Press, Menlo Park, CA.

Resende, M. and Ribeiro, C. (2002) Greedy randomized adaptive search procedures. In *Handbook of Metaheuristics*, F. Glover and G. Kochenberger (Eds.), Kluwer, Dordrecht, pp. 219–249.

Tsang, E. (1993) *Foundation of Constraint Satisfaction.* Academic, London.

Verfaillie, G. and Lemaitre, M. (2001) Selecting and scheduling observations for agile satellites: some lessons from the constraint reasoning community point of view. In *Principles and Practice of Constraint Programming, 7th International Conference, CP 2001*, Lecture Notes in Computer Science, Vol, 2239, T. Walsh (Ed.), Springer, Berlin, pp. 670–684.

Scheduling the Internet

TOWARDS AN XML-BASED STANDARD FOR TIMETABLING PROBLEMS: TTML

Ender Özcan
Yeditepe University, Istanbul, Turkey
eozcan@cse.yeditepe.edu.tr

Abstract A variety of approaches have been developed by researchers to solve different instances of timetabling problems. In these studies different data formats are used to represent a timetabling problem instance and its solution, causing difficulties in the evaluation and comparison of approaches and sharing of data. In this paper, a model for timetabling problems and a new XML data format for them based on MathML is proposed.

Keywords: timetabling, standard data format, scheduling, XML, MathML.

1. INTRODUCTION

Timetabling problems consist in feasible assignment of time-slots to a set of events, subject to a set of constraints. The timetabling problem is an NP complete problem (Even *et al.*, 1976). There are a growing number of solutions to different types of timetabling problems having different types of constraints (Abramson *et al.*, 1999; Alkan and Özcan, 2003; Causmaecker *et al.*, 2002; Burke *et al.*, 2003; Burke *et al.*, 1994; Burke *et al.*, 1996; Burke *et al.*, 1997; Colorni *et al.*, 1992; Corne *et al.*, 1994; Cladeira and Rosa, 1997; Dignum *et al.*, 1995; Erben and Keppler, 1995; Hertz, 1992; Monfroglio, 1988; Özcan and Alkan, 2002; Schaerf, 1996; Schmidt and Strohlein, 1979; De Werra, 1985). Since there is no common standard on specifying a timetabling problem instance and its solution proposed by a researcher, most of the results cannot be compared and benchmarking becomes almost impossible. The proposal for a common data format for timetabling was initiated by Andrew Cumming at ICPTAT'95. Studies in the area yield a language named SSTL (Burke *et al.*, 1997; Kingston, 2001). SSTL has not become a common format as expected, possibly because it is not easy to convert existing data to SSTL. Furthermore, most of the research in timetabling is due to some practical need, causing researchers to concentrate on solving the problem at hand, and to ignore the data format.

Causmaecker *et al.* (2002) argue that the timetabling research community can benefit from Semantic Web, focusing the timetabling ontology, rather than one of the layers of the architecture that requires definition of an Extensible Markup Language (XML). XML lets users create their own set of tags, enabling them to specify the structure of their documents. Furthermore, XML can be used to define a set of grammar rules to define markup languages. It is an efficient way of representing data on the web as a basis for machine to machine communication. XML documents can be considered to be a globally linked database. There are already defined XML-based languages. For example, MathML provides a means to use mathematical expressions on the web; Scalable Vector Graphics (SVG) is a language for describing two-dimensional graphics in XML. Details about technologies related to XML can be found on the W3C site (W3C, 2004).

Timetabling problems can be formulated using set theory as described in Section 3, where a constraint is a function operating on the sets. Hence, MathML provides a basis for the representation of timetabling components. For example, MathML allows users to define completely new content symbols that represent a function or a type or another content markup element. This important feature can be used to standardise some timetabling constraints, providing flexibility for users to define their own constraints as well.

In this paper, Timetabling Markup Language (TTML), an XML-based data format for timetabling problems, is presented utilising MathML content markup.

2. TTML: TIMETABLING MARKUP LANGUAGE

It is vital to clearly define and represent the elements of a timetabling problem using TTML. The same requirements explained in previous works will be considered during the process (Burke *et al.*, 1997; Kingston, 2001).

This section is an overview of TTML tags for content markup to generate a well-formed document. For a world-wide accepted format for representing timetabling problems, a working group should come together under W3C from researchers and vendors. TTML will be developed further whether this action is taken or not. Instead of creating a new approach, TTML extends MathML, intensifying the importance of modelling. The elements of TTML are built around MathML. The aim is to address the underlying issues, and come up with possible solutions during the modelling. Note that the conversion between different XML documents with similar contents is easy and this conversion does not require a valid document. Hence, XML Schema is left as a further study. All bold TTML elements are optional elements, "|" denotes or and "[ ]" denotes one or more occurrence of the element enclosed.

2.1 MathML

MathML is an XML based standard for describing mathematical expressions (W3C, 2004). Presentation markup defines a particular rendering for an expression, while content markup in the MathML provides a precise encoding of the essential mathematical structure of an expression. Some of the content markup elements include relations, calculus and vector calculus, theory of sets, sequences and series, elementary classical functions and statistics. Note that `declare` element is a MathML constructor for associating default attribute values and values with mathematical objects. In TTML, `declare` is used to associate a name with the defined sets. Attributes `desc` and name are proposed for `declare` element in TTML, denoting a short description of the declared item and a unique name associated with it, respectively. Unless mentioned otherwise, the order of TTML elements are strict.

2.2 Modelling Timetabling Problem Instances

An XML document requires one unique root element. The root element is chosen to be `time-tabling` for a timetabling problem instance. Our first aim should be enabling data exchange; hence a TTML document must include input data and the constraints for the problem instance. Additionally, for the research community, in order to make comparisons, test results obtained from applying an algorithm to the input data should be attached. Further attachments might be required, such as output formats for the solution. For example, course section meeting schedules can be generated as a solution to a timetabling problem, but both schedules of all teachers and students can be required as an output. Hence, a TTML document might declare multiple output formats for the same solution. Main and first level of child elements of a TTML document are illustrated in Figure 1(a). A TTML document can include an *output* format or *test results*, optionally. Element `time-tabling` can have attributes such as last update, problem type (e.g., *university course timetabling*, *highschool timetabling*, *exam timetabling*, *employee shift timetabling*), etc.

3. MODELLING INPUT DATA

Timetabling problems are constraint optimisation problems that can be represented using (V, L, C), forming input data, where $V = \{v_1, v_2, \ldots, v_i, \ldots, v_P\}$ is a set of *variables*, $L = \{d_1, d_2, \ldots, d_i, \ldots, d_P\}$, is a nonempty set of *domains* of variables, defining the set of possible values for each variable in V, and C is a set of *constraints*, where each constraint is defined for some subsets of the variables, specifying the allowable combinations of values for it. This 3-tuple forms the input data for a timetabling problem.

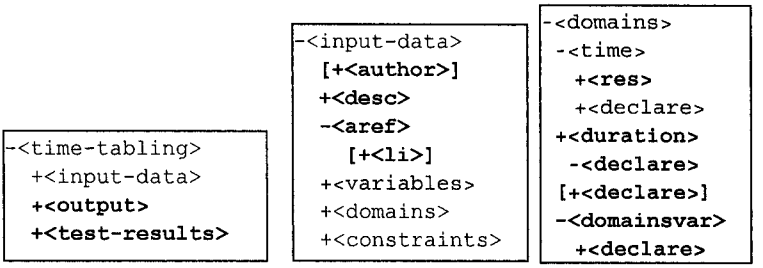


Figure 1. Main and lower level of child elements of a TTML document.

In a timetabling problem, a list or a higher dimensional array of attributes (properties) belonging to a set might be required. Let *attributed set* indicate a set where all the members have attributes. For example, the number of students registered to each course, the distances between classrooms, capacity of the classrooms can be considered as attributes of the related sets. Then the set of courses and the set of classrooms are attributed sets. Note that attribute values might be used while defining the constraints. Considering timetabling problems, we can limit the domain of the attribute values to $\mathbb{R}$, $\mathbb{Z}$, and $\mathbb{S}$.

The main and first level of child elements of input data are illustrated in Figure 1(b). Elements `author`, `desc` and `aref` describe the author of the timetabling input data, a brief description of the problem and associated references (which might be more than one), respectively. Element `variables` contains a declaration of a single set of variables, identifying each member (Figure 2(a)).

3.1 Attributed Sets

Two approaches can be applied to support attributed sets, so that attribute values could be entered as input data. Assuming that higher dimensions can be mapped into a single dimension, a `vector` can be associated with each set member as shown in Figure 2(b). If there are more than one set of attributes, then it is not straightforward how to distinguish between them, since they will all be together in the attribute vector. So, TTML shall support the second approach, allowing declaration of single or higher dimensions of attributes associated with the set using `attrset` as shown in Figure 2(c).

Element `attrset` contains two or more declarations. First declaration is a set, and the rest of the declarations (at least one) are the related attributes of it. Attribute declarations must contain `vector` or `matrix` elements where each attribute value will be accessible via `selector` function in MathML (Fig-

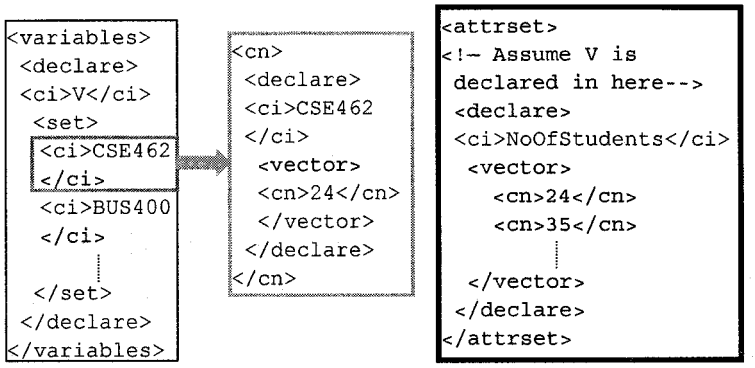


Figure 2. Possible approaches to support attributed sets as input data in TTML.

ure 2(c)). All set declarations can be replaced by attributed set declarations in TTML. If the element `attrset` is used to represent a set, then although the first declaration is a set, the order of the elements in the set becomes important, and for the rest of the attribute declarations, that order will be used to identify an attribute value. For example, 24 is the number of registered students. Note that an attributed set might contain more than one set of attributes. It is assumed that the corresponding attribute value of a set member is associated by keeping the same order, wherever the member is defined. For example, assuming CSE462 is the first member of the attributed set then the *NoOfStudents* attribute value of it will be the first entry of the corresponding declaration, which is 24 as shown in Figure 2. Thus one defines the `attrval` element, accepting the attributed set member and attribute set name as input and returning the attribute value of an attribute set member.

4. MODELLING DOMAINS OF VARIABLES

A candidate solution V' of a timetabling problem is defined by an assignment of values from the domain to the variables:

$$V' = \{v_1 = v'_1, \ldots, v_i = v'_i, \ldots, v_p = v'_p\}$$

where $v'_i \in d_i$ and

$$d_i \subseteq D_1 \times D_2 \times \cdots \times D_1 \times \cdots \times D_n, \quad 1 \leq n \tag{1}$$

While defining the constraints this assignment function might be needed. For this reason, a TTML element `assignvar` is defined as a unary function requiring a single argument which must be a member of the set of variables, returning

the assigned value. For example, `assignvar`(v_i) returns v_i'. A domain consists of either time intervals (*time set*) or Cartesian product of several sets, one of them being the time set ($D_1 = T$). If a domain is a Cartesian product of multiple sets then `domainsvar` element should be used for domain declaration in TTML. In such a case, the assignment might be an n-tuple, represented by a `vector` in TTML, allowing access of any *dimension* via `selector` function. For example, assuming a set of courses as a set of variables, `assignvar`(v_i) might return $(4, A200)$, indicating an assignment of the ith course to the fourth time interval in the timetable which will meet in the classroom $A200$. Selecting the first element in the vector returns 4; the second element returns $A200$.

It is possible that in some timetabling problems, durations might be also in the domain of a variable. TTML shall support declaration of duration set using `duration` element. Each member of duration set must be of type duration as explained in the following section. All of the related sets must be *declared* in a TTML document as domains of variables as shown in Figure 1(c).

In timetabling problems, a timetable is either discrete or continuous. In TTML, a combination of both is also supported for generality. *Time intervals* in a timetable might have equal or unequal length, or be periodic or non-periodic. In some problems date, in some others date and time, might be required. TTML shall support all.

4.1 Modelling Time Interval and Duration

A time interval can be represented using a starting time and a *duration*. MathML does not contain any type definition related to time or duration, but it allows user-defined types. Similar definitions for `dateTime` and `duration` types in XML schema are proposed to describe a time interval in TTML. In order to get rid of the confusion and be able to use a total order on time, Coordinated Universal Time (UTC) is chosen using the syntax CCYY-MM-DDThh:mm:ss. Duration type syntax is PnYnMnDTnHnMnS, indicating the number (n) of years (Y), months (M), and so on. Any substring generated using the syntaxes defined above will be valid, assuming that the string includes at least one time item. Duration set as a domain of a variable will be composed of members that are of duration type. This set shall be bound to a name using a declaration as shown in Figure 1(c), if used. TTML shall support three functions; `tistart`, `tiduration` and `tiend` returning the starting time, duration and end of a time interval, requiring a single parameter.

4.2 Modelling Timetables

A user should be able to define his/her own formatting string, emphasising the time elements relevant to the problem and then the timetable. In TTML, `res` element will be used to state the format of the time used in the timetable

definition. For example, the quantity `10-10T10:00 <sep/> P1H` represents a time interval on the 10th day of October with duration 1 hour, based on the formatting string `<res>MM-DDThh:mm</res>`. A timetable is, ultimately, a set of time intervals. TTML shall support this most general approach, enabling shortcuts. An attribute, named as `interval` is added to the `set` element to identify, whether the time intervals are *continuous* or *discrete*. For example, the time set in Figure 3(a) identifies a discrete timetable with four time intervals, where on the first day, the first interval starts at 10 with 50 minute duration, the second one starts at 11 with 50 minute duration, and on the second day, the first interval starts at 10 with 50 minute duration, and the second one starts at 11 with 50 minute duration. A timetable can be assumed to be a two-dimensional structure, as the name suggests. We can consider that this structure contains a number of columns. Each column element is ordered within itself and each column is ordered as well, providing a total order on time. Three functions are proposed for defining a timetable as a domain of variables: `spread`, `spreadcolumn`, and `tmatrix`. Usage of these functions is illustrated in Figures 3(b)–(d). The *spreadcolumn* function repeats a given set of time intervals for a given number of times by interleaving a given duration in between them and returns a time set (Figure 3(b)). The *spread* function repeats a given time interval for a given number of times, forming a column, and then applies *spreadcolumn*, using a given interleave and repetition (Figure 3(c)).

In some discrete timetabling problems, instead of time intervals, indices indicating a timetable slot can be used. Function *tmatrix* generates a discrete timetable of a given number of rows and columns, in which each timetable slot is identified by its row and column index and time line proceeds in column major order on the matrix generated (Figure 3(c)). Both *spreadcolumn* and *spread* own *interval* attribute indicating whether the timetable is *discrete* or *continuous*. Furthermore, in the discrete case, constraints will refer to timetable slots using start times for the corresponding time interval, by default. It has been observed that time slots in a discrete timetable might also be referred to using two indices, their row and column indices, or using a single index, while defining the constraints.

For example, Figures 3(a)–(c) describes the timetable illustrated in Figure 4. Ignoring the dashed lines, Figure 3(d) identifies the very same table. There are three ways to refer to the marked time slot in the timetable: 2T10, (1,2), 3 or 2. Single indices 3 and 2 are produced by a column major and row major scan on the timetable, respectively. TTML shall support row-column and column major order single indexing for referrals during constraint declarations other than the default. For this reason, for all table defining functions having discrete intervals, an additional attribute named `itype` is proposed, indicating the type of the indexing mechanism to be used for timetable slots. The supported val-

```
<time>
 <declare>
 <ci>T</ci>
  <set
    interval="discrete">
   <cn>1T10<sep/>P50M</cn>
   <cn>1T11<sep/>P50M</cn>
   <cn>2T10<sep/>P50M</cn>
   <cn>2T11<sep/>P50M</cn>
  </set>
 <declare>
</time>
```

(a)

```
<apply>
  <spreadcolumn    interval=
"discrete"/><ci><set>
    <cn>1T10<sep/>P50M</cn>
    <cn>1T11<sep/>P50M</cn>
   </set> </ci>
  <!-- interleave -->
  <cn>P1D</cn>
  <!-- repetition -->
    <cn>2</cn>
</apply>
```

(b)

```
<apply>
  <!-column major order-->
  <tmatrix/>
  <cn>2</cn><!--row-->
   <cn>2 </cn><!--column-->
<apply>
```

(d)

```
<apply>
 <spread
   interval="discrete"/>
  <cn>1T10<sep/>P50M</cn>
  <cn>P1H</cn>
  <cn>2</cn>
  <cn>P1D</cn>
  <cn>2</cn> </apply>
```

(c)

Figure 3. Defining a timetable in TTML.

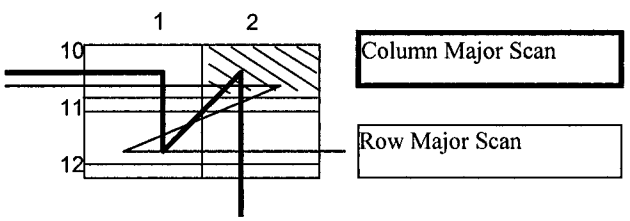


Figure 4. Timetable described in Figure 3 and indexing mechanisms using a single index.

ues are `default`, `row-column`, and `column-major`. Depending on the *itype*, TTML shall allow a user to identify the start index to be $(0,0)$ or $(1,1)$ for *row-column*, or 0 or 1 for *column-major*. The `start` attribute belonging to table defining functions will take value of either 0 or 1.

5. MODELLING CONSTRAINTS

Constraints are classified as *hard* and *soft* for timetabling problems. Hard constraints are the most common constraints. Soft constraints are the preferences that are strongly desired. In general, six different constraint types can be identified for TTPs: *edge constraints, exclusions, presets, ordering constraints, event-spread constraint* and *attribute constraints* (includes *capacity constraints*). Details about these constraints can be found in Fang (1994).

The problem of determining the minimum number of time slots needed subject to some basic constraints (edge constraints, exclusions, presets), is a graph colouring problem, studied by many researchers (Leighton, 1979; De Werra, 1985). Constraints are functions to be applied on variables or subsets of variables or their related attributes. Since MathML supports user-defined functions, *constraints* in TTML are proposed to be *declaration* of *functions* grouped as *hard/soft*.

Example Assume that we have two sets of courses; ES and CS and it is required that no pair of variables should be scheduled at the same time, where each pair is an element of the Cartesian product of ES and CS.

Pairing up all the events that should not overlap and using that as input data would not be practical, yet a feature that should be supported in TTML. Instead, while defining the constraint function, the sets in question can be used directly and computation of the Cartesian product of sets can be supported by TTML, possibly as in Figure 5. Being a function, each constraint requires parameters in TTML. Hence, TTML should allow users to define subsets of variables via *classifiers*, representing logical groupings in a hierarchical way. In this way, the user will be able to use the same constraint function for different sets defined in the same TTML document.

Define a *classifier* to be a set which is either a subset of variables, named as *base classifier*, or a set of classifiers. Notice that classifiers can form a hierarchy, just like rooted trees. For this reason a *similar* terminology will be used. A *parent classifier* is a classifier having non-base classifiers as members. Each member of a parent classifier is called *child classifier*. By default, the variables set forms a base classifier that should not be redeclared. Revisiting the example, ES and CS classifiers can be defined as base classifiers, and then the constraint function in Figure 5 would be supported.

In TTML, before the constraint functions are defined, classifiers that will be used in the constraint functions must be declared. Element `set` is used to declare child and base classifiers in a recursive manner. Element `rootcl` is used to declare root classifiers only. Each set is bound to a name using the `declare` element. A parent classifier might contain a classifier that is already defined. TTML should avoid redeclarations of the *same* classifiers.

```
<apply>
  <forall/> <bvar> <ci> x </ci> <ci> y </ci> </bvar>
  <condition>
    <apply>  <and/>
     <apply> <in/><ci> x </ci><ci> ES </ci> </apply>
     <apply> <in/><ci> y </ci><ci> CS </ci> </apply>
    <apply/>
  </condition>
  <apply> <neq/>
  <ci><apply> <selector/><ci>
    <apply><assignvar/> <ci> x </ci></apply> <ci/>
    <cn>1<cn/></apply></ci>
  <ci><apply> <selector/><ci>
    <apply><assignvar/> <ci> y </ci></apply><ci/>
    <cn>1<cn/></apply></ci>
  </apply>
</apply>
```

Figure 5. A constraint function imposing that no two events, one from ES and the other from CS sets, should overlap, assuming a discrete timetable.

Considering all the above concerns, the `constraints` element is designed as illustrated in Figure 6.

Additionally, a function is needed to convert a parent classifier into a subset of variables. For example, assume that we have two base classifiers, one identifying courses with laboratories (ESL), the other identifying courses without labs (ESN) with ES code and ES being a parent classifier such that ES = {ESL, ESN}. Assume the same for the courses in CS: CS = {CSL, CSN}. Then the constraint function in Figure 5 cannot be applied on ES and CS. A union of all the members of base classifiers of ES and CS should be generated. Define *self-projection* of a parent classifier to be a base classifier, generated by applying union on each member classifier recursively down to the base classifiers. Hence applying self-projection on ES and CS would return the expected arguments for the constraint function in Figure 5. Define *child projection* of a parent classifier to be a set of base classifiers, generated by applying self-projection on each member classifier recursively down to the base classifiers. As an example, applying child projection on a parent classifier ALL, defined as ALL = {ES, CS}, would return a two-member set of self-projections of ES and CS. TTML shall support self-projection using the `self-project` element and child projection using the `child-project` element, requiring a single argument that is a parent classifier.

The constraints part includes a set of constraint items, indicated by `coi` element, where each item can be defined as a function with or without binding

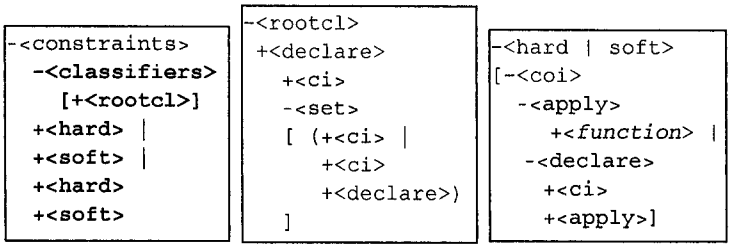


Figure 6. Main and the lower level child elements of constraints, where `function` element represents a specific constraint function.

of the returned result to an identifier using the *declare* element. The reason that the user is allowed to bind the result to an identifier is to support a sequential filtering mechanism. The output of applying a constraint can be fed into another constraint as an input. The output of a constraint is discussed in the following section.

5.1 TTML Constraint Functions

Leaving all the constraints to be defined by the user might lead to some problems during data sharing. For example, two different ways can be used to define the semantically same function. For this reason, some basic standard functions should be identified covering at least most of the constraint types. This would also reduce the effort of writing all well known constraint functions in MathML.

It is assumed that constraint functions are control functions checking some conditions and with the elements that do not satisfy a constraint being decided by the problem solver. Table 1 displays the standard constraint functions supported by TTML. Functions O0–O5 return the set of variables (or pairs of variables) that do not satisfy the related constraint. O5 returns all the pairs of variables that do not satisfy the event spread constraint along with the real gap between two events.

Functions O6 and O7 are used to define how a group of events should be distributed over the timetable. Function O6 is used for determining the distribution of filled slots, while O7 is used for determining the distribution of empty slots between filled slots due to a set of events. For example, a user might impose a constraint of a workload for a student per day. Note that the workload might be required to be distributed to a whole week, or the workload might be required to exclude the days when a student does not have any courses. Also, in a student course schedule, minimum empty slots might be required between

ID	Functions	Explanation	Semantic
00	<notsame/> • v_i • v_k	v_i and v_k are not same	$\texttt{assignvar}^{\texttt{dim}}(v_i) \neq \texttt{assignvar}^{\texttt{dim}}(v_k)$
01	<nooverlap/> • v_i • v_k	No overlap between v_i and v_k	$\texttt{tiend}(\texttt{assignvar}^{\texttt{t}}(vi)) \leq \texttt{tistart}(\texttt{assignvar}^{\texttt{t}}(v_k))$
02	<preset/> • v_i • S	Include the set S as the domain of v_i	$\texttt{assignvar}^{\texttt{t}}(v_i) \in S$
03	<exclude/> • v_i • S	Exclude the set S from the domain of v_i	$\texttt{assignvar}^{\texttt{t}}(v_i) \notin S$
04	<ordering comp=">|<|="/> • v_i • v_k	v_i is after (smaller) | before (larger) | same as (equal to) v_k	$\texttt{assignvar}^{\texttt{t}}(v_i)(> \mid < \mid =) \texttt{assignvar}^{\texttt{t}}(v_k)$
05	< eventspr comp = " < | > | = | ≤ | ≥ "/ > • v_i • v_k • d	The difference between v_i and v_k must be less than | greater than | equal to | greater than or equal to | less than or equal to d	$\texttt{tiend}(\texttt{assignvar}^{\texttt{t}}(v_i))+ d(> \mid < \mid = \mid \leq \mid \geq) \texttt{tistart}(\texttt{assignvar}^{\texttt{t}}(v_k))$
06	< fullspr per = "duration" all = "yes|no" comp = " > | < | = |avr | ≤ | ≥ "/ > • V_i • d	The total number of assignments of each variable in set V_i per *duration* throughout the whole timetable has to be greater than | less than | equal to | on average | less than or equal to | greater than or equal to d (if the interval contains any assignment (all=no))	
07	< freespr per = "duration" block = "on|off" all = "yes|no" comp = " > | < | = |av r| ≤ | ≥ "/ > • V_i • d	The total number of empty slots between each variable assignment (assuming consecutive assignments as single block of assignment, if block = on) in set V_i per *duration* throughout the whole timetable has to be greater than | less than | equal to | on average | less than or equal to l greater than or equal to d (if the interval contains any assignment (all = no))	

ID	Functions	Explanation	Semantic
O8	< attrcomp comp = " > | < | = | ≤ | ≥ "/ > • v_i • a • p • b • r	Compares the selected attribute p value of the variable v_i along a defined dimension a and selected attribute r value of the assignment along a defined dimension b	attrval (assignvar$^{a}(v_i), p$) (> | < | = | ≤ | ≥) attrval(assignvar$^{b}(v_i), r$)
O9	<resnoclash/> • v_i • v_k	If the assignments of the selected dimension (domain) are same for a pair of variables, then there must be no overlap between the time assignment of v_i and v_k,	If assignvar$^{dim}(v_i)$) == assignvar$^{dim}(v_k)$) then tiend(assignvar$^{t}(v_i)$) ≤ tistart(assignvar$^{t}(v_k)$)
O10	< chksum per = "duration" tt = "common| separate" comp = " > | < | = |av r| ≤ | ≥ "/ > • V_i • d • r	Forms a data structure where each entry spans time slots of the timetable *duration* long. If r is *default* and tt is *common*, then the function scans assignment of all the elements in V_i and using the timetable mappings of each entry, it increments the related field in the data structure, After the scan is complete, quantity at each field is *compared* with d, using the selected criterion. If r is a selected attribute, then the quantity in a field is incremented by the corresponding attribute value of an element in V_i. Setting tt to *separate* creates a data structure for each member classifier in V_i.	

Table 1. Functions assuming that assignvar$^{t}(v_i)$< assignvar$^{t}(v_k)$, where v_i and v_k are variables and assuming that t represents the time dimension of the assignment.

course meeting blocks. O6 and O7 return a positive real value as compared to d.

Function O8 is for comparing attribute values. For example, the number of students taking a course should not exceed the capacity of a classroom. The number of student is an attribute of a variable, while capacity is an attribute of a classroom. The O8 function is supported for such constraint declarations, returning the variables that do not satisfy the related constraint.

The O9 function checks whether two assigned values along a dimension are the same or not. If they have the same value, then it checks for time overlap. This function is for scheduling resources, other than the ones variables represent, without a clash. For example, the constraint imposing that the courses

should not be scheduled to the same classrooms at the same time can be defined using O9 (Figure 7(c)). The O9 function returns all the pairs of variables that do not satisfy the constraint.

The O10 function returns an array having an equal size with the timetable divided by the duration, where each entry is an aggregation of a selected quantity at a group of timetable slots determined by the duration. An entry of the array is a pair. One of the pairs is the absolute difference between the total sum and the entered value and the other is a Boolean flag indicating the comparison result. For example, in a final exam timetabling problem, a schedule disallowing 1500 students to be seated at the same time might be required. O10 is a useful construct to define such constraints.

For no overlap, we know that comparison is made with a pair of time interval items, so no dimension selection is needed, even if the timetabling problem involves a Cartesian product of multiple sets as domains of variables. But in such a case, for other functions, the dimension should be selected using the `dim` attribute, where the corresponding value should be either an index or a name, indicating a domain set, otherwise the n-tuple resulting from `assignvar` should be used. Except for the O1 and O6–O8 functions, all the rest have the `dim` attribute in TTML. O9 cannot have *time* as a value of its *dim* attribute and the O8 function accepts dimension as an input. O1, O6 and O7 can be used to define only time-related constraints on a variable set, whereas the rest can be used to define constraints on a selected dimension (domain).

There are three input cases for O1–O5, other than single events:

1 A binary function accepting a single set

2 A binary function accepting two sets

3 A unary function accepting a single set

For these cases, self-projections of the input sets will be taken, and then the related function will be applied on the resulting base classifier. For example, *nooverlap* function can accept a base classifier as input, indicating that no pair in the set should overlap, or it could accept a parent classifier which will be reduced to a base classifier by self-projection. Case 1 can be enriched by several more interpretations. Single set parameter might be a parent classifier and the user would like to apply the binary function on any pair in each child projection. For binary and unary functions accepting sets as their parameters, attribute `projection` is proposed with values "`single|self|child`", indicating a base classifier, self-projection of a parent classifier or a child projection of a parent classifier, respectively. Note that O6, O7 and O10 do not accept a single variable as their parameter. Using this feature, the function definition in Figure 5 reduces to the function call in the constraint declaration in Figure 7(a).

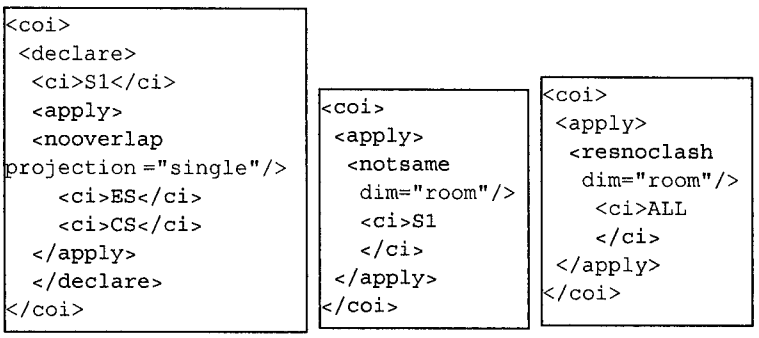


Figure 7. Declaration of a constraint as a function (a), a filtering example (b), the same effect of (b) requiring no filtering, where ALL = ES U CS (c).

A filtering example is illustrated in Figure 7(b). It is more appropriate to use standard functions, although the same affect can be obtained using filtering as shown in Figures 7(a)–(c).

6. MODELLING OUTPUT AND TEST RESULTS

Modelling output is necessary for a general tool to be able to generate expected visual timetables for a given timetabling problem instance. For example, the schedules of each instructor or/and the schedule of students belonging to the same set can be asked to be produced. In the schedule, the user might prefer to see the classroom assignments as well. For this reason, the output element should allow the display of *listed* items, where each item is a different view of variable assignments. Each *li* is assigned `info`, an attribute indicating the assigned values of dimensions that will be printed out into the timetable slot. More than one dimension can be referred to in *info*, separated by commas. Furthermore, a user might expect to see the classroom schedules. For this reason, *info* can get `variables` as an attribute value. Then, selecting a classroom in *li* and *variables* an *info* attribute value will yield a timetable output of the selected classroom schedule.

Each item can be a single variable, a set of variables (a base classifier) or sets of set of variables (a parent classifier). If it is a set of variables, the user might require an output for each variable (`each`), or a single output for all variables (`all`). If it is a parent classifier, the user might require an output for each variable in the set of self-projection (`each`), or a single output for all variables in the set of self-projection (`all`), or an output for each child

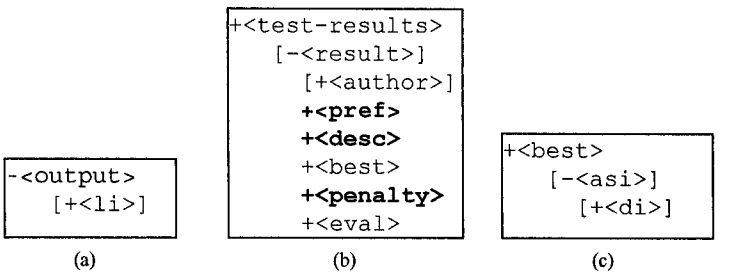


Figure 8. The main and the lower level child elements of (a) output, (b) test results, (c) best assignment of variables.

projection (`child`). For this reason, another attribute is supported to be used with *li*, which is `gentype`, requiring one of the values `each`, `all` or `child`.

Test results are for researchers, containing the best assignment of variables, author of the test results, references, a short description about the problem solver (algorithm), the best result obtained and the evaluation function. Each assignment of a variable is marked by `asi` element. The order of each assignment is assumed to be in the order of how variables are defined. Since each assignment might consist of several values, depending on the domains of variables, each domain item is included inside the element `di`. The main and the lower level child elements of output, test result and the best assignment of variables are shown in Figures 8(a)–(c), respectively.

In most timetabling applications, *penalising* an unsatisfied constraint is traditional. For supporting related evaluation functions TTML allows an optional declaration of a penalty value using `penalty` element for each defined constraint.

7. CONCLUSIONS

TTML can model all real-world timetabling problems based on MathML. In some situations, the user might be required to use some non-standard variable, domain and constraint declarations. TTML is not a widely accepted standard, but using TTML with standard constraint functions, most of the university course timetabling, highschool timetabling, final exam timetabling and some of the shift timetabling problems can be modelled. TTML requires standardisation of more constraint functions to cover more of the employee timetabling problem instances.

TTML is designed to include even the test results for comparison. For the time being, test results consist of the best results compiled from different tests. This definition can be modified to include more information for each test on the runs, such as statistics of number of evaluations, statistics of timings, or

properties of the machine on which the experiments are performed, etc. A full TTML document can be retrieved from a web site by an expert application for timetabling. This application can perform experiments on the given data subject to given constraints, and then compare its results with the best results obtained previously. Furthermore, the application can update the TTML document using its best results by modifying the test results part of the retrieved TTML document. TTML requires more work in modelling the evaluation function, additional to determining more standard constraint functions.

TTML provides all the advantages and strengths of XML. Applications can be carried to the Internet, becoming services. A *TTML processor* can be designed having three major components: a parser, problem solver and a solution interpreter. A *multipurpose* TTML processor is the ultimate goal that solves different types of timetabling problems. Using TTML, data sharing will be easy and fast. Additionally, TTML provides means to include different parts of other TTML documents in order to make use of previously defined components and their features (variables, constraints, etc), using Xlink and Xpath technologies (W3C, 2004), where the same functionality is provided by SSTL using an object-oriented methodology. The requirements for a standard data format can be summarised as universality (assuming a closed world), completeness and convertibility. The latter requirement is satisfied by TTML, just by being an XML standard. TTML, powered by MathML, is a strong candidate for satisfying all these requirements.

Furthermore, recent studies concentrate on case-based reasoning approaches, which can benefit from the use of TTML. Burke *et al.* (2003) define similarity measures to support such approaches. In some problems, it is important to determine the strongly connected components (possibly the largest one) in a graph, mapping timetabling problem into a graph colouring problem. Finding maximal clique is an NP complete problem. If TTML is used and the variables are grouped into classifiers, then the problem of determining the strongly connected components reduces to locating classifier sets as parameters of a related constraint.

A TTML validator is not implemented, since there are many general validators available over the Internet. Figure 10 includes a well-formed TTML document. The first TTML application, named CONFETI, is implemented as a Java applet, providing a user interface to convert final exam timetabling text data into a TTML document. CONFETI will be used to build an instance repository providing TTML documents for final exam timetabling using existing data. Initially, Carter's benchmark data sets are converted to TTML, based on the constraints defined in Burke *et al.* (1996), successfully. The second application will be available soon: a full TTML processor based on a memetic algorithm for final exam timetabling, named FES (Final Exam Scheduler). The results will be reported soon. The latest developments in TTML

```
<?xml version="1.0"?>
<time-tabling>
   <input-data type="University-Course-Timetabling"
   lastUpdate="2003-01-22T13:20:00.000-05:00">
   <author>Ender Ozcan</author>
   <desc>An example TTML document</desc>
   <variables>
     <attrset>
      <declare> <ci>V</ci>
      <vector>
          <ci duration="2"> CSE211.01</ci>
          <ci duration="2"> CSE211.02</ci>
          <ci> CSE311.01</ci>
          <ci> CSE462.01</ci>
      </vector>
      </declare>
      <declare> <ci>noOfStudents</ci>
      <vector>
        <ci> 34</ci>
        <ci> 27</ci>
        <ci> 20</ci>
        <ci> 25</ci>
      </vector>
      </declare>
      </attrset>
   </variables>
   <domains>
     <time>
       <declare> <ci>T</ci>
       <apply>
         <tmatrix itype="row-column" start="1">
           <cn> 9</cn> <cn> 5</cn>
         </tmatrix>
       </apply>
       </declare>
     </time>
     <attrset>
      <declare> <ci>classrooms</ci>
        <vector>
          <ci> A100</ci>
          <ci> B101</ci>
          <ci> B103</ci>
          <ci> A201</ci>
```

```
      </vector>
    </declare>
    <declare> <ci>capacity</ci>
     <vector>
      <ci> 50</ci>
      <ci> 50</ci>
      <ci> 50</ci>
      <ci> 30</ci>
     </vector>
    </declare>
  </attrset>
  <domainsvar>
    <declare><ci>R</ci>
           <apply><cartesianproduct/>
             <ci> T </ci>
             <ci> classrooms </ci>
           </apply>
    </declare>
        </domainsvar>
  </domains>
  <constraints>
    <classifiers>
      <rootcl> <declare> <ci>lecturers</ci>
          <set> <ci> <declare> <ci> Ender Ozcan </ci>
           <set>
            <ci> CSE211.01</ci>
            <ci> CSE311.01</ci>
           </set>
          </declare> </ci>
          <ci> <declare> <ci> Ferda Dogan </ci>
           <set>
            <ci> CSE211.02</ci>
            <ci> CSE462.01</ci>
           </set>
          </declare> </ci>
      </set> </declare> </rootcl>
      <rootcl> <declare> <ci>curriculum-terms</ci>
          <set> <ci> <declare> <ci> Term#2 </ci>
          <set>
           <ci> CSE211.01</ci>
           <ci> CSE211.02</ci>
           <ci> CSE311.01</ci>
          </set>
         </declare> </ci>
```

```
        <ci> <declare> <ci> Term#3 </ci>
         <set>
          <ci> CSE462.01</ci>
         </set>
        </declare>  </ci>
    </set> </declare> </rootcl>
  </classifiers>
  <hard>
    <coi>
      <apply>
        <nooverlap projection="child"/>
          <ci>curriculum-terms</ci>
      </apply>
    </coi>
    <coi>
      <apply>
        <nooverlap projection="child"/>
          <ci>lecturers</ci>
      </apply>
    </coi> <coi>
      <apply>
        <preset dim="classroom"/>
          <ci>CSE211.01</ci>
          <cn>A100</cn>
      </apply>
    </coi>
    <coi>
      <apply>
        <exclude dim="T"/>
          <ci>CSE311.01</ci>
          <cn><vector> <cn>1<sep/>1</cn>
              <cn>2<sep/>2</cn></vector>
          </cn>
      </apply>
    </coi>
    <coi>
     <apply>
      <resnoclash projection="single"
       dim="classrooms"/>
        <ci>V</ci>
     </apply>
    </coi>
  </hard>
```

```
      <soft>
        <coi>
        <apply>
         <attrcomp projection="single" comp="&lt;="/>
           <ci>V</ci>
           <ci>T</ci>
           <ci>noOfStudents</ci>
           <ci>classrooms</ci>
           <ci>capacity</ci>
        </apply>
        </coi>
      </soft>
    </constraints>
    </input-data>
    <output>
       <li projection="child">lecturers</li>
    </output>
    <test-results>
      <result>
         <author>Ender Ozcan</author>
         <desc>  Problem is solved by TEDI, following is
                 the best solution obtained in 50 runs
         </desc>
         <best>
           <asi> <di>4<sep/>1</di> <di>A100</di> </asi>
           <asi> <di>4<sep/>2</di> <di>B101</di> </asi>
           <asi> <di>4<sep/>3</di> <di>A201</di> </asi>
           <asi> <di>4<sep/>4</di> <di>B103</di> </asi>
         </best>
      </result>
   </test-results>
</time-tabling>
```

Figure 9. An example TTML definition of a timetabling problem instance.

and instance repositories will be available at http://cse.yeditepe.edu.tr/ eozcan/research/TTML.

References

Abramson, D., Dang, H. and Krisnamoorthy, M. (1999) Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, **16**:1–22.

Alkan, A. and Özcan, E. (2003) Memetic algorithms for timetabling. In *Proceedings of 2003 IEEE Congress on Evolutionary Computation*, pp. 1796–1802.

Causmaecker, P. D., Demeester, P., Lu, Y. and Vanden, G. (2002) Using web standards for timetabling. In *Practice and Theory of Automated Timetabling IV*, Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 238–258.

Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic S. and Qu, R. (2003) Similarity measures for exam timetabling problems. In *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, pp. 120–135.

Burke, E. K., Elliman, D. and Weare, R. (1994) A genetic algorithm based timetabling system. In *Proceedings of the 2nd East–West International Conference on Computer Technology in Education*, pp. 35–40.

Burke, E. K., Newall, J. P., Weare, R. F. (1996) A Memetic Algorithm for University Exam Timetabling. In *Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 241–250.

Burke, E. K., Pepper P. A. and Kingston, J. H. (1997) A standard data format for timetabling instances. In *Practice and Theory of Automated Timetabling II*, Lecture Notes in Computer Science, Vol. 1408, Springer, Berlin, pp. 213–222.

Colorni, A., Dorigo, M. and Maniezzo, V. (1992) A genetic algorithm to solve the timetable problem. *Technical Report* 90-060 revised, Politecnico di Milano, Italy.

Corne, D., Ross, P., Fang, H. L. (1994) Evolutionary timetabling: practice, prospects and work in progress, *Proceedings of the UK Planning and Scheduling SIG Workshop*.

Cladeira, J. P. and Rosa, A. C. (1997) School timetabling using genetic search. In *Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1408, Springer, Berlin, pp. 115–122.

Dignum, F. P. M., Nuijten, W. P. M., Janssen, L. M. A. (1995), Solving a Time Tabling Problem by Constraint Satisfaction, *Technical Report,* Eindhoven University of Technology.

Erben, W. and Keppler, J. (1995), A Genetic algorithm solving a weekly course-timetabling problem. In *Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 21–32.

Even, S., Itai, A. and Shamir, A. (1976) On the Complexity of Timetable and Multicommodity Flow Problems, *SIAM Journal of Computing*, **5**:691–703.

Fang, H. L. (1994) Genetic algorithms in timetabling and scheduling, *Ph.D. Thesis.*

Hertz, A. (1992) Finding a feasible course schedule using a tabu search, *Discrete Applied Mathematics*, **35**:255–270.

Kingston, J. H. (2001) Modelling timetabling problems with STTL. In *Practice and Theory of Automated Timetabling III*, Lecture Notes in Computer Science, Vol. 2079, Springer, Berlin, p. 309.

Leighton, F. T. (1979) A graph coloring algorithm for large scheduling problems, *Journal of Research of the National Bureau of Standards*, **84**:489–506.

Monfroglio, A. (1988) Timetabling Through a Deductive Database: A Case Study, *Data and Knowledge Engineering*, **3**:1–27.

Özcan, E. and Alkan, A. (2002) Timetabling using a steady state genetic algorithm. In *Practice and Theory of Automated Timetabling II*, Lecture Notes in Computer Science, Vol. 1408, Springer, Berlin, pp. 104–107.

Schaerf, A. (1996) Tabu Search Techniques for Large High-School Timetabling Problems, *Proceedings of the 14th National Conference on AI*, pp. 363–368.

Schmidt, G. and Strohlein, T. (1979) Time table construction—an annotated bibliography, *The Computer Journal*, **23**: 307–316.

De Werra, D. (1985) An introduction to timetabling, *European Journal of Operations Research*, **19**:151–162.

W3C (2004) World Wide Web Consortium web site: http://www.w3c.org.

A SCHEDULING WEB SERVICE

Leonilde Varela
Department of Production and Systems, University of Minho
Campus Azurem, 4800-058 Guimaraes, Portugal
leonilde@dps.uminho.pt

Joaquim Aparício
Department of Computer Science, New University of Lisbon
Quinta da Torre, 2829-516 Monte de Caparica, Portugal
jna@di.fct.unl.pt

Sílvio do Carmo Silva
Department of Production and Systems, University of Minho
Campus Gualtar, 4710-057 Braga, Portugal
scarmo@dps.uminho.pt

Abstract This paper describes a web system for supporting manufacturing scheduling in practice. Within this system scheduling concepts like problems and solving methods are modelled through XML. Furthermore, the Internet remote methods' invocation is also based on the XML language and performed using the XML-RPC communication protocol. The architecture and underlying approach of the web system is oriented for solving a large variety of manufacturing scheduling problems based on a continuously updatable distributed knowledge base, which allows a network of peers to provide the scheduling service to users and the dynamic enlargement of the number of methods that can be accessed. This is done in an easy and interactive way.

Keywords: manufacturing scheduling, XML modelling, web service and remote procedure call.

1. INTRODUCTION

The scheduling activity in an organisation seeks to optimise the use of available production means or resources, ensuring short time to complete jobs and, in addition, to satisfy other important organisation objectives. Thus, it can sig-

nificantly contribute to good service to customers and to high profitability of an organisation. Manufacturing Scheduling may be defined as the activity of allocating tasks to production resources, or vice versa, over time. The result of this is usually expressed in a production schedule.

With this work we make a contribution for the better resolution process of scheduling problems by means of a web-based system. This system requires, first of all, the specification and identification of each problem to be solved and, then, the access to resolution methods, which are available for solving them. When there are different methods available we can obtain alternative solutions, which should be evaluated against specified criteria or objectives to be reached. Thus, we are able to properly solve a problem, through the execution of one or more scheduling methods. These methods can either be local or remotely accessible through the Internet.

The remote invocation of the methods is performed by a web service through the XML-RPC (extensible markup language, remote procedure call) protocol. The web service accepts as input a problem definition, i.e. data for running a method, and returns outputs from the method's execution.

To be able to implement the web service and the other important functionalities of the web system, it is essential to specify the scheduling problems, which in our case is done by means of a scheduling problem classification model.

Manufacturing scheduling problems and related concepts, including solving method information are modelled through XML. This kind of data modelling also allows remote method invocation to enable one to establish the necessary communication for execution through the web.

This paper is organised as follows. The next section describes the scheduling situations for which this system is intended and the underlying scheduling problems classification model. Section 3 presents a general outline of the web system architecture and briefly describes the main modules and associated processes of the web system. Sections 4, 5 and 6 describe and illustrate the scheduling methods specification, the methods searching in a distributed knowledge base and the remote methods invocation processes, respectively. Section 7 briefly refers to some related work and the last section presents some conclusions.

2. MANUFACTURING SCHEDULING

Scheduling problems belong to a much broader class of combinatorial optimisation problems, which, in many cases, are hard to solve, i.e. are NP-hard problems (Ceponkus, 1999; Jordan, 1996; Blazewicz, 1996; Brucker, 1995). In the presence of NP-hard problems we may try to relax some constraints imposed on the original problem and then solve the relaxed problem. The solution of the latter may be a good approximation to the solution of the orig-

inal one. Many times, due to the short time we have to make decisions, we do not have a choice and have to draw upon what we may generally call approximation methods (French, 1982). These include both those of which we know how near their solutions may be from optimum ones and also a variety of heuristic methods. Important techniques on which many of such methods are based are dynamic programming, branch and bound techniques and meta-heuristics. Typical examples of meta-heuristics, also known as extended neighbourhood search techniques, widely used nowadays, include genetic algorithms, tabu search and simulated annealing (Osman, 1996; Arts, 1997). We should also mention other successful scheduling approaches, frequently used, based on simulation, bottleneck theory, neural networks and Petri nets, among others.

2.1 Scheduling Scenarios

We can encounter in practice, in manufacturing environments, a huge variety of scheduling problems that have to be solved. In general, this means to solve jointly or separately a set of interrelated and complementary actions or stages that the scheduling activity involves. These usually include order release, task assignment and sequencing on resources and also detailed scheduling. Here detailed information about when each task should start and finish on particular resources, main and auxiliary resources, must be established.

The web system here presented enables the selection and use of methods for solving a large variety of real-world manufacturing scheduling problems. These may be subject to several combinations of constraints related to tasks and resources. A classification model is used for specifying the wide range of scheduling problems that can occur in different manufacturing systems. This model is briefly described here.

2.2 Specification of Scheduling Problems

Due to the existence of a great variety of scheduling problems there is a need for a formal and systematic manner of problem classification that can serve as a basis for their specification. A classification model for achieving this was developed by Varela (1999) and Varela *et al.* (2002a, b), based on published work by Conway (1967), Graham *et al.* (1979), Brucker (1995), Blazewicz (1996) and Jordan (1996), as well as on other information presented by Morton (1993), Artiba (1997) and Pinedo (1995). This model allows identification of the underlying characteristics of each problem to be solved and is used as a basis for the XML-based problem specification model used in this work.

The referred classification model for problem specification includes three classes of notation parameters for each corresponding class of problem characteristics, in the form of $\alpha|\beta|\gamma$. The first class of characteristics, the α class,

Table 1. Scheduling problem characteristics.

Class	Factor	Description	Value
α	α_1	Manufacturing system type	$\emptyset$O, P, Q, R, X, O, J, F, PMPM, EJ, ...
	α_2	Number of machines	Om
β	β_1	Job preemption	$\emptyset$O, pmtn
	β_2	Precedence constraints	$\emptyset$, prec, chain, tree, sp-graph, ...
	β_3	Ready times	$\emptyset$O, r_j
	β_4	Restrictions on processing times	$\emptyset$, p_j=1, p_{ji}=1, p_j=p, $p_{inf} \leq p_j \leq p_{sup}$, ...
	β_5	Due dates (deadlines)	$\emptyset$O, d_j
	β_6	Batch	$\emptyset$O, batch$_j$
	β_7	Complex jobs	$\emptyset$, comp$_j$
	β_8	Number of jobs or tasks in a job	n
	β_9	Job priorities	$\emptyset$, w_j
	β_{10}	Machine eligibility	$\emptyset$, M_{js}
	β_{11}	Dynamic machine availability	$\emptyset$, avail$_i$
	β_{12}	Additional/auxiliary resources	$\emptyset$, aux$_k$
	β_{13}	Buffers	$\emptyset$, buffer$_{ji}$,no-wait
	β_{14}	Setup (changeover)	$\emptyset$, setup
γ	γ	Performance measure	C_{max}, F_{max}, $\sum C_j$, $\sum w_j C_j$, L_{max}, $\sum T_j$, ...

is related with the environment where the manufacturing is carried out. It specifies the manufacturing system type and, when necessary, the number of machines that exists in the system. This parameter enables definition of the main class to which scheduling problems belong. Examples of these are parallel machines, flow-shops, open-shops, job-shops, manufacturing cells, flexible systems and extended job-shops (EJ). The EJ assumes manufacturing of batches of either simple or complex products (Almeida *et al.*, 2003). The former products are made of single parts. The latter include the manufacturing of parts and their assembly into products. The β class, which includes the β_1 to β_{14} parameters, allows specification of the job and resources characteristics. Some important constraints are imposed by the need for auxiliary resources, such as tools, handling devices and/or the existence of buffers. The class γ deals with the performance criteria, which can be based on a single criterion performance measure or on multi-criteria measures. Table 1 gives an overview of this three-field problem classification model. More detailed information about this model can be found in Varela *et al.* (2002a, b).

An example of use of this notation is "F3|n,d_j, batch$_j$|Cmax" which can be read as: "Scheduling of an arbitrary number of jobs with defined due dates and batch sizes, on a pure flow-shop, with three machines, to minimise the maximum completion time". Due to the absence of a number of parameters, in

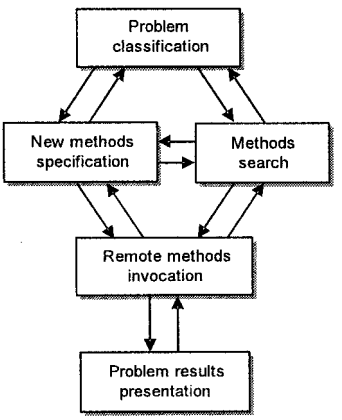


Figure 1. Main system processes.

the problem representation notation, this means that the related characteristics are either absent from the problem or specified by default, corresponding to the $\emptyset$ symbol.

3. WEB SYSTEM DESCRIPTION

This web system is a tool for aiding users to solve scheduling problems in manufacturing. Figure 1 shows the main system's processes. These processes are carried out by four main modules: a user interface module (UIM), a knowledge specification module (KSM), a method searching module (MSM) and a method invocation module (MIM). The UIM enables one to input problems and methods specification data and also to present results in different formats. The KSM enables the updating of a distributed knowledge base (DKB). The DKB is a distributed repository about scheduling knowledge, i.e. specified scheduling problems and also methods for supporting the manufacturing scheduling process. One important piece of the MSM is a Prolog-based searching mechanism for finding suitable methods for solving a given scheduling problem. The MIM module is used for remotely accessing methods to solve problems.

The application area and potential of the web scheduling system is mainly dependent on its ability to specify and deal with scheduling problems to be solved. This very much depends on the contents of the DKB and, to a certain extent, also on the ability of problems to be clearly specified into the system. This is restricted by the classification model above described. Although much effort has already been put by the authors into developing a comprehensive model, capable of allowing the specification of a large variety of problems, it

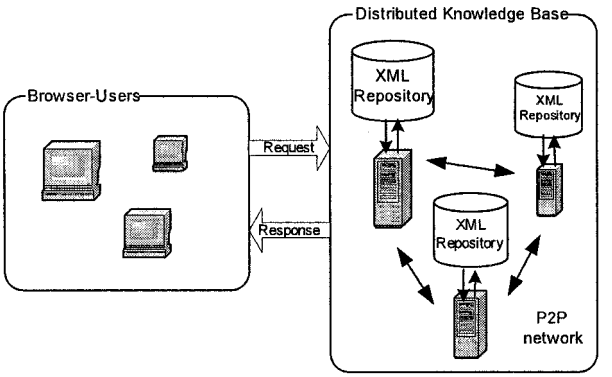


Figure 2. Web-system architecture.

can always be extended to enlarge the set of problems that can be dealt with by the system.

The contents of the DKB are critical because they provide the methods which can be identified and used for solving a specified scheduling problem.

The DKB is embedded in a peer-to-peer (P2P) network (for details see http://www.oreilly.com/catalog/peertopeer/).

The peers communicate with each other in a searching process for adequate scheduling methods to solve problems specified by the user. He or she can connect to one of the peers for finding methods, in the local knowledge base or in other peers' knowledge bases, for solving a given scheduling problem. If available, a method can be executed under request.

Figure 2 illustrates a general outline of the web system architecture.

XML was considered suitable and important to support the development of the web system here described (Varela *et al.*, 2002a, b)). Some important XML applications, also relevant to manufacturing, are PDML (Product Data Markup Language), RDF (Resource Description Format), STEPml (Harper, 2001), JDF (Job Definition Format), PSL (Process Specification Language), PIX-ML (Product Information Exchange), PIF (Process Interchange Format) and XML-based workflow (Abiteboul *et al.*, 2000).

In the knowledge base component of each peer the scheduling data are stored in XML documents and each document is validated using an associated DTD (document type definition), before being updated in the corresponding XML repository.

Elements on the DTD for problems precisely characterise scheduling problems, meaning that in order to interact with the system a problem must be described according to that grammar (Varela *et al.*, 2002a, b).

The UIM is mainly controlled by DTD and XSL (extensible stylesheet language) documents stored in the local knowledge base of each peer. The system has been designed and implemented as a web service (http://www.w3.org) and will be available soon through the http://www.dps.uminho.pt/ web site.

XML-RPC (Laurent *et al.*, 2001; Varela *et al.*, 2002) is the communication protocol that is used for remote method invocation. Other protocols could be considered, namely SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery, and Integration of business for the web), WSDL (Web Services Description Language), or other well known ones, such as CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invocation) or DCOM (Distributed Component Object Model).

4. METHODS SPECIFICATION

The DKB is easily and dynamically updatable by users, which include not only end-users willing to solve problems but also domain experts wishing to specify methods into the system. This is done using the KSM module and requires using the problem classification model, which is essential for the web system to work.

Many implementations of a given method may be accessible through the Internet. From the point of view of the web system two implementations of the same method may differ if, for example, they differ in outputs. Moreover, not all implementations work in the same way. Therefore, for the system to be able to match problem instances to resolution methods and to retrieve and use implemented methods available they must be specified into the system. This specification must include, among other things, the uniform resource locator to run a method, the communication protocol identification and the implemented method's signature, which, in turn, must include the definition of the parameters that are necessary for its invocation, i.e. the inputs, and its output format.

Figure 3 illustrates the method's signature specification of an implementation of the branch and bound method proposed by Ignall and Schrage (1965) for a flow-shop problem. The signature inputs, in this example, include the definition of a parameter n for number of jobs to be processed, which is of integer type, a parameter m for number of machines in the production system and also a set of three items in a matrix structure, which represent the job name, the machine name and an additional parameter p that corresponds to job processing time. There is also the definition for the method's output following the same lines. This information is subsequently inserted in an XML document in order to enable repeated executions of methods, information retrieval and automatic generation of interfaces for the implemented methods' inputs and outputs.

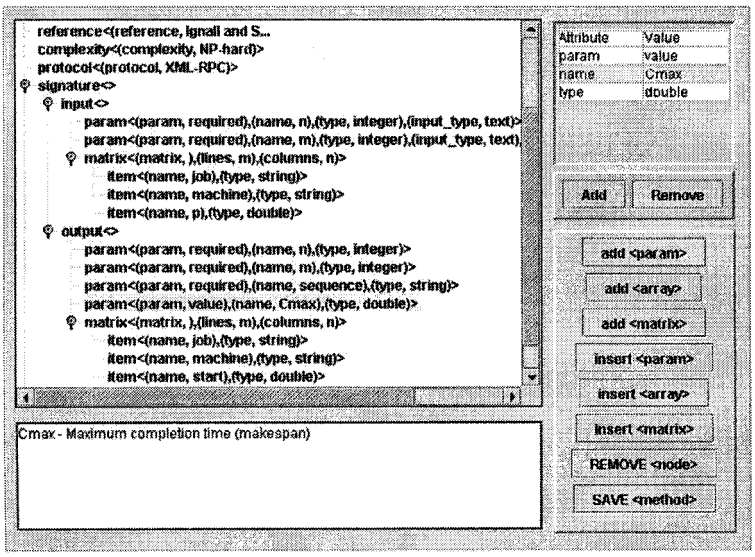


Figure 3. Method signature definition.

For the example given, after the insertion of the Ignall and Schrage's method definition, any method search that is a match for the Fm|n|Cmax problem class will include this method in the search results.

5. METHODS SEARCHING

The MSM, referred to in Section 3, for finding suitable methods for solving scheduling problems, is based on the classification model referred to in Section 2.2. The MSM includes a searching mechanism that selects, from each local knowledge base of the DKB, the scheduling method(s) that can be used for solving a specified problem. This is a built-in Prolog engine using an internal inference mechanism developed using the SWI-Prolog V.5.2.1. free software tool available at http://www.swi-prolog.org/. The searching mechanism identifies a list of problem classes from the DKB that matches the characteristics of the user-specified problem. The user is presented with this list, from which the problem to be solved can be identified.

An example of a list of problem classes, resulting from specified characteristics of a practical problem, is presented in Figure 4. The referred to characteristics can be seen on top of the figure.

After the user has selected one or more suitable problems from the list and submitted a request to search for solving methods, the system shows a list of

Problem types for the general problem characteristics: [system_type=F, jobs=n, measure=Cmax]

Class	Complexity	Context	Problem characteristics	Select
F2|n|Cmax	Maximal Polinomially Solvable	≥	[(system_type, F), (machines, 2), (jobs, n), (measure, Cmax)]	☑
F2|rj,n|Cmax	Minimal NP-hard	≥	[(system_type, F), (jobs, n), (arrivals, rj), (machines, 2), (measure, Cmax)]	☐
F2|rj,n,no-wai	Maximal Polinomially Solvable	≥	[(system_type, F), (machines, 2), (jobs, n), (arrivals, rj), (buffers, no-wait), (measure, Cmax)]	☐
F3|rj,n|Cmax	Minimal NP-hard	≥	[(system_type, F), (machines, 3), (jobs, n), (arrivals, rj), (measure, Cmax)]	☐
F3|n|Cmax	Minimal NP-hard	≥	[(system_type, F), (machines, 3), (jobs, n), (measure, Cmax)]	☑
Fm|n|Cmax	Minimal NP-hard	≥	[(system_type, F), (machines, m), (jobs, n), (measure, Cmax)]	☑
Fm|prec,pj=1,	Minimal NP-hard	≥	[(system_type, F), (machines, m), (jobs, n), (precedences, prec), (times, pj=1), (measure, Cmax)]	☐

Figure 4. Problem class selection.

Available methods for solving the problem classes: [class=F2|n|Cmax, class=F3|n|Cmax, class=Fm|n|Cmax]

ID	Prob. Class	Method name	Implementation location	Reference	Complexity	Protocol	Select
1001	Fm|n|Cmax	BranchBoundIS	http://localhost:6002/RPC2	Ignall and Schrage, 1965	NP-hard	XML-RPC	◉
32	F2|n|Cmax	Johnson	http://localhost:6002/RPC2	Johnson, 1954	Polynomial	XML-RPC	○

Figure 5. Method implementation for solving problem classes.

those referred to in the DKB. The list includes author(s) and literature reference, details for aiding on the selection of adequate scheduling methods for solving the problem and the links for method executions, if they are available.

An example of the output of the searching methods process is presented in Figure 5. As illustrated, the system has one method available, which the user selected, for solving Fm|n|Cmax problems. This method is the branch-and-bound method from Ignall and Schrage (Ignall and Schrage, 1965; Baker, 1974). Further information is available, such as, in this case, that the method belongs to the class of exact mathematical programming methods and is an exponential time complexity method. The Figure 5 also includes the Johnson method (Johnson, 1954; Baker, 1974) for solving F2|n|Cmax problems.

If no method is available in the DKB for solving a given problem, the user is encouraged to relax some of the problem constraints or introduce further problem characteristics and repeat the searching process.

6. METHODS INVOCATION

Our web service is provided through the MIM module. This, as previously referred to, performs the remote method invocation for solving manufacturing scheduling problems.

In the web service a certain method accepts as input a problem definition and returns a result in some particular form. Therefore, the system has to exactly

know the implemented methods' signatures. All this information is described in a corresponding DTD file (Varela *et al.*, 2002a, b).

Different implementations may provide results in different formats. Therefore, the system must have a description of them in order to format them according to the problem output to be returned to the client. This is the last step of the service. The result from running a method implementation on the given problem instance can then be delivered to the client as an XML file and/or can be transformed into some more expressive output, like tables or Gantt charts.

The web service uses XML to encode both the message wrapper and the content of the message body. As a result, the integration is completely independent of operating system, language or other middleware product used by each component participating in the service. The only fundamental requirement is that each component has the ability to process XML documents and that each node connected in a distributed environment supports HTTP as a default transport layer.

The XML-RPC protocol is the sequence and structure of requests and responses required to invoke communications on a remote machine. The eXtensible Markup Language provides a vocabulary for describing remote procedure calls, which are transmitted between computers using the Hyper Text Transfer Protocol (HTTP). XML-RPC clients make procedure requests to XML-RPC servers, which return results to the XML-RPC clients. XML-RPC clients use the same HTTP facilities as web browser clients, and XML-RPC servers use the same HTTP facilities as web servers. XML-RPC requires a minimal number of HTTP headers to be sent along with the XML method request for solving a given scheduling problem instance (Varela *et al.*, 2003).

For a better illustration, let us consider an instance of the previously described Fm|n|Cmax problem class: namely, a problem with four jobs, which have to be processed in a flow-shop with three machines in order to minimise the maximum completion time (Cmax), with the time required for processing each job j on each machine i, as shown in Table 2.

For solving the problem instance under consideration, which belongs to class Fm|n|Cmax, with m equal to 3 and n equal to 4, we can select the Branch and Bound method of Ignall and Schrage, shown in Figure 5, implemented in C++ and running under the XML-RPC protocol.

After having selected the implemented method for solving our problem, we only need to feed the system with the problem instance data and run it.

The problem data, as well as the information related to methods, are specified through XML; the system enables an easy automatic generation of interfaces for problem data insertion for each particular problem instance. Figure 6 illustrates the interface for the problem instance under consideration, according to the previously chosen method, whose signature is known by the system, due to its previous specification in the web system.

Table 2. Scheduling problem data.

$i\backslash j$	L1	L2	L3	L4
M1	3	11	7	10
M2	4	1	9	12
M3	10	5	13	2

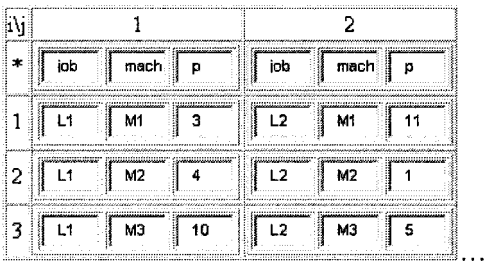


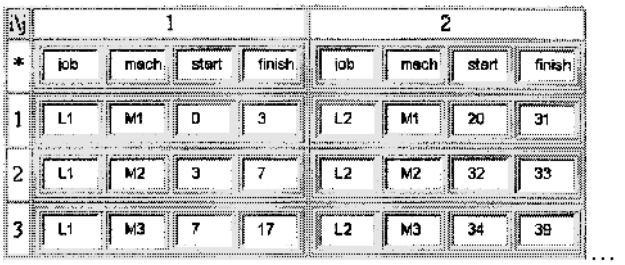
Figure 6. Interface for inserting problem data.

Upon receiving an XML-RPC request, an XML-RPC server must deliver a response to the client. The response may take one of two forms: the result of processing the method or a fault report, indicating that something has gone wrong in handling the request from the client. If everything has gone well, once having executed the implemented method, the system automatically generates the interface for presenting the results, again accordingly to the predefined method's signature. Figure 7 exemplifies the interface generated by the system for presenting the results obtained for our problem.

The system also enables another kind of result presentation, namely Gantt charts, which are also automatically generated by the system, from the outputs provided by the execution of methods. This is easily achieved because the output data are expressed in XML and XSL documents, which enables an easy method of output conversion into different presentation forms.

Figure 8 shows the Gantt chart representing the optimal solution obtained for the problem instance considered, with a completion time of 39 time units.

Gantt charts continue to be widely used due to their expressive power, giving an easy way of comparing results obtained from the execution of several different implemented methods available. Other alternatives for displaying outputs are available, including direct output presentation through XML documents.



	1				2			
*	job	mach	start	finish	job	mach	start	finish
1	L1	M1	0	3	L2	M1	20	31
2	L1	M2	3	7	L2	M2	32	33
3	L1	M3	7	17	L2	M3	34	39

Figure 7. Interface for presenting problem results.

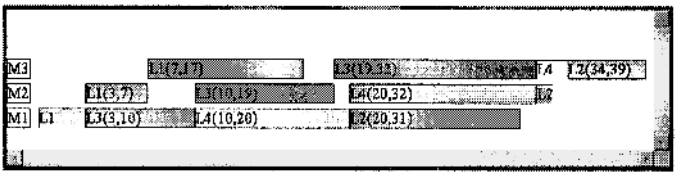


Figure 8. Gantt chart.

7. RELATED WORK

We have noticed a recent increase in scheduling systems accessible through the Internet. These usually involve solvers or a community of solvers, each one addressing the resolution of a restricted range of scheduling problems, using specific techniques or tools, such as mathematical programming. They are not usually designed to easily incorporate new method implementations by users.

An example of a web system that can be used for scheduling is the NEOS Server, developed under the auspices of the Optimisation Technology Centre of Northwestern University and Argonne National Laboratory, for optimisation problem solving. It makes nearly 50 solvers available through a broad variety of network interfaces. According to the authors, although having evolved along with the web and the Internet, it is limited to some degree by early design decisions (http://www-neos.mcs.anl.gov/).

We also mention the BBN Vishnu scheduling system, a web-based optimisation scheduling system (http://vishnu.bbn.com) and the FortMP, a Mathematical Programming Solver, from Mitra's Group at Brunel University (see http://www.brunel.ac.uk/depts/ma/research/com).

The e-OCEA, a portal for scheduling, intends to help identifying scheduling problems, to help development of new algorithms and to conduct benchmarks through the Internet. However, this system will only consider elements

(algorithms, data sets, schedules and modules) that are e-OCEA compatible (http://www.ocea.li.univ-tours.fr/eocea).

The LekiNET, a prototype Internet scheduling environment, by Benjamin *et al.* (Yen *et al.*, 2004) which is evolved from LEKIN, a flexible job shop system, is a system that has some similarities to our system, focusing more on cost effective choice of scheduling agents for solving problems. The authors propose a migration scheme to transform existing standalone scheduling systems to Internet scheduling agents that can communicate with each other and solve problems beyond individual capabilities. They treat each system as an agent and build the relations between the systems. Therefore, wrappers need to be specifically designed for each system.

In our case, any method which is accessible through the Internet, provided its signature and location are specified within our web system, can be used for solving problems put by users. No further requirements are necessary for being able to remotely use available methods. The association of scheduling problems with resolution methods is done using the information available in the DKB, about both problems and solving methods. We have not come across systems with identical architecture and underlying approach, i.e. oriented for solving a large variety of manufacturing scheduling problems based on a continuously updatable distributed knowledge base, which allows a network of peers to provide the scheduling service to users and the dynamic enlargement of the number of methods that can be accessed.

8. CONCLUSION

In manufacturing enterprises, it is, nowadays, important, as a competitive strategy, to explore and use software applications, now becoming available through the Internet and Intranets, for solving scheduling problems.

This work presents a web-based system for aiding manufacturing scheduling in practice. The essential processes available include the ability to specify scheduling problems and to find appropriate methods for solving them. This is based on a problem classification model that allows the specification of a large variety of scheduling problems that may occur in different types of real world manufacturing environments.

Manufacturing scheduling data specification using the XML language is one important aspect underlying this work. This modelling and specification contributes to the improvement of the scheduling process, by allowing an easy selection of several alternative available methods for problem solving, as well as an easy update of the distributed knowledge base that supports this web system. This knowledge primarily includes scheduling problems and resolution methods. The methods may be implemented on different programming lan-

guages and running on different platforms and can be easily invoked through this web system, when available through the Internet.

As a prototype, some methods are accessible through the web system allowing comparison of different solutions obtained by different methods for solving each given scheduling problem.

Although the main goal is the service for scheduling problem solving, the system is also expected to be used for teaching purposes, and from this point of view, some additional features are being introduced into the system. These include historical and referencing information about each problem class and solving method. The system will grow as more knowledge is included in the distributed knowledge repository.

References

Abiteboul, S., Buneman, P. and Suciu, D. (2000) *Data on the Web—From Relations to Semistructured Data and XML*, Morgan Kaufmann, San Mateo, CA.

Almeida, A. Ramos C. and Silva, S. C. (2003) Product Oriented Scheduling through Job Scheduling Patterns, *Proceedings of International Conference on Industrial Engineering and Production Management* (IEPM'03, Porto, Portugal).

Artiba, A. and Elmaghraby, S. (1997) *The Planning and Scheduling of Production Systems*, Chapman and Hall, London.

Arts, E. and Lenstra, J. K. (1997) *Local Search in Combinatorial Optimization*, Wiley, New York.

Baker, K (1974) *Introduction to Sequencing and Scheduling*, Wiley, New York.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. and Weglarz, J. (1996) *Scheduling Computer and Manufacturing Processes*, Springer, Berlin.

Brucker, P. (1995) *Scheduling Algorithms*, Springer, Berlin.

Ceponkus, A. and Hoodbhoy, F. (1999) *Applied XML*, Wiley, New York.

Conway, R. W., Maxwell, W. L., and Miller, L. W. (1967) *Theory of Scheduling*, Addison-Wesley, London.

French, S. (1982) *Sequencing and Scheduling—An Introduction to Mathematics of the Job-Shop*, Wiley, New York.

Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, **5**:287–326.

Harper, F. (2001) *XML Standards and Tools*, Excelon Corp., USA.

Ignall, E. and Schrage L. (1965) Application of the branch-and-bound technique to some flow-shop problems, *Operations Research*, **13**:400–412.

Jordan, C. (1996) *Batching and Scheduling*, Springer, Berlin.

St Laurent, S., Johnston, J. and Dumbill, E. (2001) *Programming Web Services with XML-RPC*. O'Reilly, Cambridge, MA.

Morton, T. and Pentico, D. (1993) *Heuristic Scheduling Systems*, Wiley, New York.

Osman, I. H. and Kelly, J. P. (1996) *Meta-Heuristics: Theory and Applications*, Kluwer, Dordrecht.

Pinedo, M. (1995) *Scheduling Theory, Algorithms and Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Varela, L., Aparício, J. and Silva, S. (2003) A scheduling web service based on XML-RPC. In *Proceedings of the Doctoral Consortium*, Unger H. *et al.*, ICAPS'03 Conference (Trento, Italy) NASA Ames, USA.

Varela, L., Aparício, J. and Silva, S. (2002a) An XML knowledge base system for scheduling problems. In *Proceedings of the Innovative Internet Computing Systems*, by Unger H. *et al.*, during the Innovative Internet Computing System Conference (Kuhlungsborn, Germany), Lecture Notes in Computer Science, Springer, Berlin.

Varela, L., Aparício, J. and Silva, S. (2002b) Scheduling problems modeling with XML. In *Proceedings of Research in Logistics*, J. C. Carvalho *et al.*, 4th International Meeting for Research in Logistics (Lisbon, Portugal).

Varela, M. L. R. (1999) Automatic scheduling algorithms selection, *M.Sc. Dissertation*, Department of Production and Systems, University of Minho, Portugal. .

Yen, B. and Wu, O. (2004) Internet scheduling environment with market-driven agents. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **34**(2).

Machine Scheduling

AN O($N \log N$) STABLE ALGORITHM FOR IMMEDIATE SELECTIONS ADJUSTMENTS

Laurent Péridy and David Rivreau
Institut de Mathématiques Appliquées, Université Catholique de l'Ouest, France
{ laurent.peridy,david.rivreau } @ima.uco.fr

Abstract Using local operations within branch-and-bound methods for job-shop scheduling problems has been proved to be very effective. In this paper, we present an efficient algorithm that applies ascendant set-like adjustments for the immediate selections. This procedure is given within an original framework that guarantees a good convergence process and an easy integration of other classical disjunctive elimination rules.

Keywords: disjunctive scheduling, edge finding, local adjustments, elimination rule, job-shop.

INTRODUCTION

Many scheduling problems found in a typical factory environment involve the processing of jobs on a fixed set of machines that can handle at most one job at a time. If we focus on one machine, we are given a set of operations to be processed without interruption in their time windows. The purpose of local adjustments is to narrow these time windows in order to speed up the enumerative approaches used for the whole problem. This kind of elimination rule has been in particular successfully applied to solve to optimality notoriously difficult scheduling problems such as job-shops (Carlier and Pinson, 1989; Brinkkötter and Brucker, 2001). In this paper, we consider the immediate selections due to Carlier (1975) and give an $O(n \log n)$ procedure that finds all adjustments associated with these selections.

The paper is organised as follows. In the first section, we recall the main classical adjustment procedures and give some properties that entitle to design stable algorithms. Section 2 is devoted to the presentation of the new elimination rule. Then, in Section 3, this procedure is implemented by a stable procedure that it is proved to run in $O(n \log n)$ time. Finally, we report some experimental results on job-shop in Section 4 and draw some conclusions in Section 5.

1. LOCAL ADJUSTMENTS FOR DISJUNCTIVE PROBLEM

1.1 Disjunctive Scheduling Problem

As mentioned before, we concentrate on the process of a set $\mathcal{O}$ of n operations on a single machine that can process only one operation at a time. Each operation i from $\mathcal{O}$ is given an integer processing time p_i and must be processed in a certain time window $[r_i, d_i]$. No pre-emption is allowed. Therefore, any feasible schedule of $\mathcal{O}$ is characterised by a set $\{t_i\}$ of starting times for operations such that the following two relations hold:

$$r_i \leq t_i \leq d_i - p_i \qquad \forall i \in \mathcal{O}$$
$$(t_i + p_i \leq t_j) \vee (t_j + p_j \leq t_i) \qquad \forall (i, j) \in \mathcal{O} \times \mathcal{O}$$

The main goal of local operations is precisely to reduce the time windows bounds of operations in order to reduce the problem size. Since adjustments of release dates and of deadlines are clearly symmetrical, we will henceforth only consider release date adjustments.

1.2 Local Adjustments

One of the first local adjustments has been proposed by Carlier (1975). This elimination rule attempts to deduce an adjustment from the relative positioning of two given operations i and j. It can be stated as follows:

> *Immediate selections adjustments (Carlier, 1975).* If $r_j + p_j + p_i > d_i$ then i precedes j in any feasible solution. In that case, we can let
>
> $$r_j \leftarrow \max(r_j, r_i + p_i)$$

These immediate selections have been extended by Carlier and Pinson (1989). To this end, they evaluate the relative positioning of an operation i in a given subset J such that $i \notin J$. Three cases are distinguished:

(C1) Operation i cannot be scheduled *before* subset J if

$$r_i + p_i + \sum_{j \in J} p_j > \max_{j \in J} d_j$$

(C2) Operation i cannot be scheduled *inside* subset J if

$$\min_{j \in J} r_j + p_i + \sum_{j \in J} p_j > \max_{j \in J} d_j$$

(C3) Operation i cannot be scheduled *after* subset J if

$$\min_{j \in J} r_j + \sum_{j \in J} p_j + p_i > d_i$$

Carlier and Pinson deduce the so-called ascendant sets adjustments from those conditions:

> *Ascendant sets adjustments (Carlier and Pinson, 1989).* If (C1) and (C2) are satisfied then i is processed after all operations from J in any solution. In that case, we can let
>
> $$r_i \leftarrow \max \left(r_i \, ; \, \max_{J' \subseteq J} \left\{ \min_{j \in J'} r_j + \sum_{j \in J'} p_j \right\} \right)$$

It can be noticed that the potential ascendant set adjustment of r_i corresponds to the optimal makespan of the pre-emptive schedule of J.

It has been proved by Carlier and Pinson (1990, Theorem 1) that the ascendant set adjustment of i leads to the immediate selection $r_j + p_j + p_i > d_i$ for all $j \in J$. However, some of the precedence relations identified by immediate selections cannot be found by the ascendant sets procedure. It follows that a better adjustment is missed, even when ascendant sets adjustments are used with classical immediate selections.

In the remainder of the paper, we present a procedure that allows us to apply the ascendant set adjustments to all the precedence relations found by immediate selections and, by extension, induced by Carlier and Pinson (1990, Theorem 1), to all precedence relations found by the ascendant sets procedure. To distinguish our immediate selection adjustments from the original version of Carlier, we speak from now on of *improved* immediate selection adjustments.

1.3 Properties

We recall the main concepts given in Péridy and Rivreau (2005) to qualify the properties of local operations and related algorithms. In particular, we focus on the characteristics that allow us to define a class of methods for which several adjustments of release dates can be combined in a single stable pass.

So, let E be the set of n-dimensional vectors of possible release dates for a given one-machine problem. Clearly, any local adjustment can be seen as a function f from E to E. A few questions arise naturally. First of all, is it necessary to apply a local adjustment procedure in several runs to reach the fixpoint of f (in other words, does the local adjustment procedure is stable or not)? How to combine several local adjustment procedures? In what order? These questions have been investigated in Péridy and Rivreau (2005) for the

classical adjustment procedures, but in this paper, we are only interested in the first question, since we specifically focus on a single adjustment rule. Nevertheless, it remains the case that improved immediate selections adjustments are easy to integrate in the more general framework defined in Péridy and Rivreau (2005).

The stability of our general framework is based on two properties of the underlying adjustments:

- the adjustments must be *increasing*;
- the adjustments must be *non-anticipative*.

The increasing characteristic is a property defined on the following partial order $\preceq$ on E (which defines $(E, \preceq)$ as a lattice):

$$u \preceq v \quad \Leftrightarrow \quad \forall i \in \mathcal{O}, u_i \leq v_i$$

Increasing property (monotonicity). A function f from E to E—or a local adjustment—will be said to be increasing if the following relation holds:

$$\forall (u, v) \in E \times E, \quad u \preceq v \Rightarrow f(u) \preceq f(v)$$

This monotonicity characteristic is crucial to reach a unique fix-point when several adjustment procedures are involved. For more details, see Tarski (1955) and Péridy and Rivreau (2005). There is also a second, more interesting, outcome to monotonicity due to the fact that adjustments of release dates can only occurs at specific point of the planning horizon: clearly, with this property you can "jump" from two consecutive critical time breakpoints without checking the in-between values. These points—the so-called *critical time breakpoints*—are defined more precisely in the next section and roughly correspond to the initial release dates and to the completion times of some specific sets. Finally, this increasing characteristic seems a priori to be a natural property: finding less information from a more constrained problem is a little bit counter-intuitive. However, if the great majority of local adjustments are indeed monotonic, it should be noted that some of them—for instance Fix Triple Arcs (Brucker *et al.*, 1994)—are non-increasing.

Non-anticipative property. A local adjustment f is said to be non-anticipative if the final adjustment value α_i of any release date is independent of the final adjustment values of release dates of operations such that $\alpha_j \geq \alpha_i$.

This second core property means that the final adjustment α_i of initial release date r_i is only a function of all processing times, all deadlines and of final adjustments values α_j of operations such that $\alpha_j < \alpha_i$. This characteristic allows in particular a chronological study of critical time breakpoints: at each

time breakpoint we can check if a given release date reaches its final adjustment value or not. Moreover, since this value does not rely on future adjusted release dates, the overall procedure can be proved to be stable ($f \circ f = f$).

Not-first, immediate selections and ascendant set adjustments have been in particular shown to satisfy these properties (Péridy and Rivreau, 2005). In this paper, this framework is completed with the improved immediate selections.

2. IMPROVED IMMEDIATE SELECTIONS ADJUSTMENTS

2.1 Object

For the sake of clarity, the ascendant set-like adjustments for immediate selections will be precisely stated as follows:

> *Improved Immediate selections adjustments.* Let $i \in \mathcal{O}$ and also $J = \{j \in \mathcal{O} \setminus \{i\} \mid r_i + p_i + p_j > d_j\}$. Operation i must be processed after all operations from J. Hence, we can let

$$r_i \leftarrow \max\left(r_i \,;\, \max_{J' \subseteq J}\left\{\min_{j \in J'} r_j + \sum_{j \in J'} p_j\right\}\right)$$

One can observe that these improved immediate selections adjustments are increasing and non-anticipative. Indeed, increasing the value of a release date can only add new selections and also result in an increase of the values of adjustments made. Hence, improved immediate selections are increasing. Moreover, once the adjustments are stabilised, we necessarily have for any operation i:

$$\alpha_i \leftarrow \max\left(r_i \,;\, \max_{J' \subseteq J}\left\{\min_{j \in J'} \alpha_j + \sum_{j \in J'} p_j\right\}\right)$$

Since all durations are positive, it follows that any final adjustment value α_i of a release date only relies on the final adjustment values α_j of operations such that $\alpha_j < \alpha_i$, and thus improved immediate selections are also non-anticipative.

As already mentioned, these properties correspond to the framework of Péridy and Rivreau (2005): therefore, we can use here the same technique which consists in a chronological study of potential adjustment dates (the critical time breakpoints).

The present contribution will mostly concern

- the *quality* (value) of adjustments performed;
- the *stability* of the algorithm;

- the $O(n \log n)$ *complexity* of this procedure.

However, we should add that there still remains an important open question: is it possible to design an effective and stable algorithm that is able to simultaneously perform adjustments of release and due dates? Indeed, like most adjustment procedures in the literature, when the adjustment of release dates is performed it is assumed that the due dates are fixed (and vice versa). Therefore, if we consider the whole process, which implies adjusting both release and due dates, any adjustment of a due date requires us to start again the adjustment procedure on release dates (and reciprocally). Finally, it appears that the overall stability is probably a difficult problem to handle if we consider the literature, which remains very discrete on that particular subject.

2.2 Notation and Basic Properties

In order to explain and justify our procedure, we need to introduce some auxiliary notation and exhibit some properties. In the following sections we will assume that operations are numbered in increasing $d_i - p_i$ order, i.e.

$$d_1 - p_1 \leq d_2 - p_2 \leq \cdots \leq d_n - p_n$$

Let us recall that our algorithm proceeds by a chronological examination of critical time breakpoints at which adjustments can occur. For each critical breakpoint t, some operations can either reach their final adjustment value, or be delayed (i.e. adjusted on a later date). We will denote by D the set of operations that are at least delayed up to t (those who satisfy $r_i < t \leq \alpha_i$) and by L the set of operations that are not available before t (with $t \leq r_i$). Please note that for operations from $L \cup D$ we will necessarily have $\alpha_i \geq t$ at the end of the algorithm, and that operations that do not belong to $L \cup D$ have been necessarily adjusted before t. For reasons of convenience, at a given critical time breakpoint t, the α_i-values of unfixed operations—those in $L \cup D$—are arbitrarily set to $+\infty$.

Now, let us consider a given subset of operations at time t. We denote

$$C(J) = \max_{J' \subset J} \left\{ \min_{j \in J'} \alpha_j + \sum_{j \in J'} p_j \right\}$$

By definition, if J contains one operation from $L \cup D$, then $C(J)$ is arbitrarily set to $+\infty$. We will also denote by K_l the following set:

$$K_l = \{k \in \mathcal{O} \mid k \leq l\} = \{k \in \mathcal{O} \mid d_k - p_k \leq d_l - p_l\}$$

We can now express the improved immediate selections with this notation. Let us assume that we are at a given critical time breakpoint t. We need to

evaluate for operations in D and those in L with $r_j = t$, if they must be delayed or, on the contrary, if their final adjustment value α_j is equal to t. Let us denote by j a given operation from $D \cup \{k \in L \mid r_k = t\}$ and let operation i be defined as follows:

$$i = \min \{l \in \mathcal{O} \mid C(K_l) > t\}$$

Clearly, for all $k < i$, we have $k \notin L \cup D$ (otherwise, $C(K_k) = +\infty$, which is in contradiction with the definition of i). In other words, i is the only one operation from K_i that can be in $L \cup D$.

If $j \neq i$ then clearly j must be delayed if $t + p_j > d_i - p_i$. Indeed, in that case we have $t + p_j > d_k - p_k$ for all k in K_i. Hence, K_i is a valid set of predecessors for j. Since the completion time $C(K_i)$ of K_i is greater than t, then j should be delayed. On the other hand, if $t + p_j \leq d_i - p_i$, then any potential set K of predecessors is strictly included in K_i. By construction of K_i, we have necessarily $C(K) \leq t$: it follows that j cannot be delayed at time t, with respect to the improved immediate selections.

So, let us assume now that $j = i$. Clearly, operation i cannot be in the set of its potential predecessors. So we must remove i to this set and define i' as

$$i' = \min \{l \in \mathcal{O} \mid C(K_l \setminus \{i\}) > t\}$$

The same reasoning as used for j applies, and thus, we conclude that operation i must be delayed if and only if $t + p_i > d_{i'} - p_{i'}$. For $i \notin L \cup D$, we will arbitrarily set $i' = i + 1$, so in any case we have $i' > i$. Please note that if $L \cup D = \{i\}$—in other words, if $i = n$—then operation i cannot be delayed by any operation at time t (indeed, we have $C(\mathcal{O} \setminus \{i\}) \leq t$). In that case operation i' is not considered.

There is a strong relation between sets K_i and $K_{i'}$ that guarantees we avoid any removal of operation from these sets during the execution of the algorithm. Therefore, the sequence of sets K_i and $K_{i'}$ will always be increasing for the inclusion operator. This property is stated in the next proposition.

Proposition 1 *Let i and i' be defined as above for a given critical time period t. Then, we have*

$$C(K_{i'-1}) = C(K_i)$$

Proof. If $i \notin L \cup D$, the result is straightforward since $i' = i + 1$. Now, if $i \in L \cup D$, we have $C(K_{i'-1} \setminus \{i\}) \leq t$ by definition of i'. Since operation i belongs to $L \cup D$, we have $\alpha_i \geq t$. It follows that the value of $C(K_{i'-1})$ is given by the completion time of i, and $C(K_{i'-1}) = C(K_i)$. □

2.3 Example

Before describing the details of the algorithm, we will illustrate its operating mode and main characteristics through the following example.

	1	2	3	4
r_i	6	0	4	14
p_i	7	2	7	4
d_i	18	15	22	26
$d_i - p_i$	11	13	15	22

As already mentioned, we proceed by a chronological examination of critical time breakpoints that correspond to initial release dates and potential final adjustment values. For a given critical breakpoint t, operations i and i', and sets L and D are defined as in the previous section.

At every critical time, the next potential adjustment date for operations of $(L \cup D) \setminus \{i\}$ is given by $C(K_i)$. If the final adjustment value α_i has not yet been determined (if $i \in L \cup D$), it is also necessary to take into account its possible adjustment date $C(K_{i'} \setminus \{i\})$.

For brevity purposes, we start our presentation at time $t = 6$: final adjustment values α_2 and α_3 for operations 2 and 3 have been already determined to be equal to the initial release dates (since $r_2 + p_2 \leq d_1 - p_1$ and $r_3 + p_3 \leq d_1 - p_1$).

Operation i is equal to 1 and $C(K_i) = +\infty$ since operation i still belongs to L. The related operation i' is 3 because $C(K_2 \setminus \{1\}) = 2 \leq 6$. Since operation 3 has been adjusted, the exact $C(K_3 \setminus \{1\})$-value is known and is equal to 11.

Critical time breakpoint $t = 6$.

- Operation 1 becomes available: we have $t + p_1 \leq d_{i'} - p_{i'}$, then 1 is not delayed, and $\alpha_1 = 6$
- α_i is fixed: we can determine $C(K_i) = 13$. Operation i is adjusted, so $C(K_{i'} \setminus \{i\})$-value becomes useless: we set $C(K_{i'} \setminus \{i\}) = +\infty$
- The next critical time breakpoint is given by the minimum value over the release dates of operations from L and the $C(K_i)$-value: so $t = 13$.

Critical time breakpoint $t = 13$.

- $C(K_i) = t$: i and i' must be updated. From Proposition 1, we know that the next i-value is necessarily greater or equal than the current value of i'. So, we have $i \geq 3$. Moreover $C(K_3) = 18 > t$, it follows that $i = 3$. Once i updated, we need to reevaluate i'. Necessarily, i' is greater than the new i-value. Since operation 4 belongs to $L \cup D$, we deduce $i' = 4$ and $C(K_{i'} \setminus \{i\}) = +\infty$.
- The next critical time breakpoint is given by the minimum value over the release dates of operations from L and the $C(K_i)$- and $C(K_{i'} \setminus \{i\})$-values: so $t = 14$.

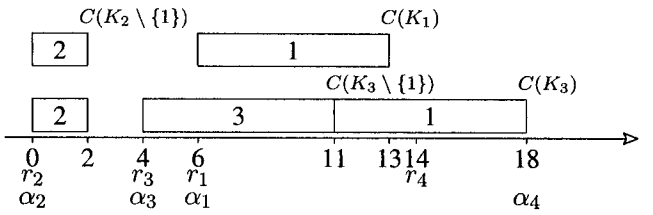


Figure 1. Example 1.

Critical time breakpoint $t = 14$**.**

- Operation 4 becomes available: $t + p_4 > d_i - p_i$, so 4 is delayed: $D \leftarrow D \cup \{4\}$
- The next critical time breakpoint is given by the minimum value over the release dates of operations from L and the $C(K_i)$-values: so $t = 18$.

Critical time breakpoint $t = 18$**.**

- $C(K_i) = t$: save operation $i' = 4$, no operation allows to increase $C(K_i)$-value, so the release date of operation 4 is definitively adjusted to $t = 18$.
- All operations are considered, the algorithm is completed.

3. IMPLEMENTATION

3.1 Notation and Basic Properties

In our algorithm, we reuse the main notation given in Section 2.2. For implementation reasons, two sets K and K' related to K_i and $K_{i'} \setminus \{i\}$ are introduced. The $C(K_i)$ and $C(K_{i'} \setminus \{i\})$ values are abbreviated in C and C'. Sets K and K' are implemented by means of red–black trees in order to be able to get in constant time the $C(K)$-value and to insert a new operation in $O(\log n)$. Please note that i and i' are inserted in K and K' only when final values α_i and $\alpha_{i'}$ are known.

As said before, the property described in Section 2.2 is the basis of the efficiency of our algorithm, since it guarantees that sets K and K' can be updated in an incremental fashion, without any removal of operation. Indeed, when K must be updated (that is when i needs to be incremented), we know that all operations between the current values of i and i' must be added to K, since we have $C(K_{i'-1}) = C(K_i)$. In more precise terms, if we note i_k, i_{k+1}, i'_k, i'_{k+1} the consecutive values of i and i' during the execution of the procedure, we have

$$i_k < i'_k \leq i_{k+1} < i'_{k+1}$$

Thus, this property enables us to gradually add operations in K and K' when necessary, that it is to say each time i or i' reaches its final adjustment value.

Finally, in the algorithm, it is implicitly assumed that there is a dummy operation $n+1 \notin \mathcal{O}$, with the following characteristics: $d_{n+1} = d_n$, $p_n = 0$.

3.2 Algorithm

The main algorithm is detailed below. At the beginning, all the operations are still to be considered and the first critical time breakpoint is the minimum release date (lines 1–2). In the main loop, we are given a current time breakpoint t.

For this critical time breakpoint, it is necessary to determine the relevant operations i and i', the related K and K' sets and C and C' values (procedure **update_i_i'**, line 4). With this information we can evaluate if the new available operation or previously delayed ones must be delayed or not (procedure **update_L_D**, line 5). After this step, all delayed operations at time t are in D.

If operation i reaches its final adjustment value at time t, it is inserted in red–black tree K and we deduce the exact value of $C(K_i)$ (line 7). Since operation i cannot be adjusted any more, this operation is also inserted in red–black tree K' (line 8). In the same way, if i' reaches its final adjustment value, it is added in K' and $C(K_{i'})$ is updated (line 10).

At last, the next critical time breakpoint to consider is updated, according to the fact that operation i can still be adjusted to C' (line 12) or not (line 13). The main loop is finished when all operations have been considered ($L \cup D = \varnothing$).

```
    procedure adjustments(r, α)
    {
1.      L ← O,   D ← ∅,   t ← min_{j∈L} r_j
2.      i ← 1, K ← ∅, C ← +∞, i' ← 2, K' ← ∅, C' ← +∞
3.      while (L ∪ D ≠ ∅) do
        {
4.          update_i_i'(t, L, D, i, K, C, i', K', C')
5.          update_L_D(t, L, D, i, i')
6.          if (α_i = t)
                then
                {
7.                  C ← insert(K, i)
8.                  insert(K', i)
                }
9.          if (α_{i'} = t)
10.             then C' ← insert(K', i')
11.         if (i ∈ L ∪ D)
```

```
12.             then t' ← min{min_{j∈L} r_j ; C ; C'}
13.             else t' ← min{min_{j∈L} r_j ; C}
        }
    }
```

Procedure **update_i_i'** is reproduced below.

First, we consider operation i and related set K: if $C = t$ then the set K cannot delay any operation after t. Therefore, it is necessary to increase i to add operations in this set (procedure **updateC**) until we get either a new operation i which has not yet reach its final adjustment value, or a definitively adjusted operation such that $C(K_i) > t$. In both cases, operations between $i' + 1$ and $i - 1$ are added in the set K'. If the new operation i is not yet adjusted (lines 3–6), we need to evaluate the new related i'. For that purpose, a call to **updateC** beginning at index $i + 1$ is made (lines 5–6). On the other hand, if i is already adjusted, then i' is not necessary for the current operation: in that case, we add i in K' for further computations, we set $i' = i + 1$ and insert i' in K' if $\alpha_{i'}$ is known (lines 7–10).

The same modus operandi is used to update operation i' (lines 11–12).

```
    procedure update_i_i'(t, L, D, i, K, C, i', K', C')
    {
1.      if (C = t)
            then
            {
2.              updateC(C, i, K, t, L, D)
3.              if (i ∈ L ∪ D)
                    then
                    {
4.                      forall k ∈ [i' + 1; i − 1] do insert(K', k)
5.                      i' ← i
6.                      updateC(C', i', K', t, L, D)
                    }
                    else
                    {
7.                      forall k ∈ [i' + 1; i] do insert(K', k)
8.                      i' ← i + 1
9.                      if (i' ∉ L ∪ D) and (i' ≠ n + 1)
10.                         then insert(K', i')
                    }
            }
11.         if (i ∈ L ∪ D) and (C' = t)
12.             then updateC(C', i', K', t, L, D)
```

```
}
```

The code of procedure **updateC** is basic: operations are inserted in the given set—in fact a red–black tree—in increasing order of $d_i - p_i$, until we get either an operation which is not yet adjusted or a set with a completion time strictly greater than t.

```
procedure updateC(Completion, index, Set, t, L, D)
{
    do
    {
        index ← index + 1
        if (index ∈ L ∪ D) or (index = n + 1)
            then Completion ← +∞
            else Completion ← insert(Set, index)
    }
    while (Completion ≤ t)
}
```

Finally, procedure **update_L_D** is also easy to state. Please note that sets L and D are implemented as heap data structures: function **top** returns—without removal—the operation with minimum release date for L and with minimum processing time for D.

In the first place, operations that have been delayed to t are considered (lines 1–8): all operation j from D that is not selected in respect to i—such that $t + p_j \leq d_i - p_i$—is removed from D, since it reaches its final adjustment value at t (lines 2–5). If i was previously delayed, we check if this operation is still selected in respect to i'. If it is not the case, operation i also reaches its final adjustment value (lines 6–8).

In the same way, operations from L that become available at time t may either be delayed t (line 13) or simply not adjusted (line 14).

```
    procedure update_L_D(t, L, D, i, i')
    {
1.      if (D ≠ ∅) then j ← top(D)
2.      while (D ≠ ∅) and (t + p_j ≤ d_i − p_i)
        {
3.          remove(D, j)
4.          α_j ← t
5.          if (D ≠ ∅) then j ← top(D)
        }
```

```
6.      if (i ∈ D) and (t + p_i ≤ d_i' − p_i')
            then
            {
7.              remove(D, i)
8.              α_i ← t
            }
9.      if (L ≠ ∅) then j ← top(L)
10.     while (L ≠ ∅) and (r_j = t)
        {
11.         remove(L, j)
12.         if ((j ≠ i) and (t + p_j ≤ d_i − p_i)) or ((j = i) and
            (t + p_j ≤ d_i' − p_i'))
13.             then insert(D, j)
14.             else α_j ← r_j
15.         if (L ≠ ∅) then j ← top(L)
        }
    }
```

3.3 Proofs

Proposition 2 *Algorithm* ***adjustments*** *is a stable procedure that performs improved immediate selections adjustments.*

Proof. As mentioned in Section 2, improved immediate selections adjustments are monotonic and non-anticipative. This means that any increase of a release date value necessarily induces better adjustments (monotonicity) and that the final adjustment value of any adjustment is only based only previously adjustments made (non anticipation). These properties allow to focus on the chronological study of potential adjustment dates (which correspond to initial release dates, C-values for all operations except operation i and C'-value for operation i) without having to test the in-between values or to go back on earlier decisions. Since the C- and C'-values correspond to the makespan of sets K_i and $K_{i'}$ as defined in Section 2.2, we deduce that **adjustments** procedure is a stable procedure that performs improved immediate selections adjustments. □

Proposition 3 *Algorithm* ***adjustments*** *runs in* $O(n \log n)$ *time.*

Proof. As mentioned above, the critical time breakpoints correspond to the initial release dates and the potential adjustments dates C and C' that are given by the makespan of sets K and K'. These sets are implemented by mean of red–black trees. In the Appendix, it is shown that **insert** procedure runs in $O(\log n)$ time. These sets only strictly increase during the algorithm, so the overall complexity to insert at most n operations is $O(n \log n)$. In the same way, each operation is inserted and removed at most once in sets D and L.

Clearly, **insert** and **remove** procedures can be done in $O(\log n)$ by mean of a heap data structure. In consequence, the overall complexity for algorithm **adjustments** is $O(n \log n)$. □

4. COMPUTATIONAL EXPERIMENTS

To evaluate the efficiency of our procedure, we performed to kind of test for our procedure: *qualitative* tests to check if improved immediate selection can contribute to get more information, and *performance* tests to see if this procedure can compete with less sophisticated ones.

4.1 Qualitative Test

In table 1, we give some results on classical job-shop scheduling problems. LB1 and LB2 are obtained by bisection search on the value of the makespan until no infeasibility is derived. LB1 corresponds to classical immediate selections whereas LB2 is related to improved immediate selections. Lower bounds LB3 and LB4 are also obtained by bisection search, but global operations (Carlier and Pinson, 1994)—also called shaving (Martin and Shmoys, 1996)—are performed respectively on classical and improved immediate selections.

One can observe that improved immediate selections clearly outperform classical immediate selections in terms of pruning.

4.2 Performance Test

In order to see if the fact that our procedure is stable and in $O(n \log n)$ counterbalance its use of a time consuming red–black tree, we compare the relative performance of two algorithms that both perform improved immediate selection adjustments: the first one is the $O(n \log n)$ stable algorithm proposed in this paper and the second one is a basic non-stable $O(n^2)$ algorithm.

Table 2 reports a comparison between the CPU times obtained for lower bounds given by shaving on 10×10 job shop scheduling problems from the literature. The first column "Stable version" is given as the reference. Table 2 shows that on average the $O(n \log n)$ stable algorithm can favourably be compared with the non-stable one. It seems that the stable algorithm especially allows one to avoid large deviations on non-stable instances.

5. CONCLUSION

In this paper, we have introduced a stable algorithm that improves immediate selections adjustments. This led to a stable procedure that can adjust release dates with these local operations in a single pass in $O(n \log n)$. Computational results confirm that this new algorithm outperforms the classical one in adjustments with comparable CPU time. This algorithm can be introduced

Table 1. Lower bound on hard instances of job-shop scheduling problems

Instance	$C^\star$	n	m	LB1	LB2	LB3	LB4
FT10	930	10	10	750	813	857	880
ABZ5	1234	10	10	975	1083	1138	1183
ABZ6	943	10	10	832	860	918	941
La16	945	10	10	810	817	890	924
La18	848	10	10	725	782	848	848
La19	842	10	10	729	751	823	825
La20	902	10	10	836	843	886	896
La21	1046	15	10	816	934	895	1004
La22	927	15	10	703	800	832	913
La24	935	15	10	787	857	860	905
La25	977	15	10	815	856	900	936
La29	1152	20	10	821	979	923	1114
La36	1268	15	15	1048	1138	1171	1234
La38	1196	15	15	1022	1065	1100	1125
La39	1233	15	15	1039	1137	1137	1221
La40	1222	15	15	1025	1115	1103	1192
ORB01	1059	10	10	792	834	919	971
ORB02	888	10	10	727	785	851	871
ORB03	1005	10	10	760	794	871	923
ORB04	1005	10	10	838	876	957	973
ORB05	887	10	10	695	746	807	840
ORB06	1010	10	10	807	838	913	951
ORB07	397	10	10	335	346	381	397
ORB08	899	10	10	676	690	773	894
ORB09	934	10	10	751	784	874	912
ORB10	944	10	10	777	855	887	930

in the framework given in (Péridy and Rivreau, 2005): indeed improved immediate selection adjustments are proved to be monotonic and non-anticipative. Since all precedence relations found by ascendant sets adjustments are selected by ascendant sets adjustments (from Theorem 1 in (Carlier and Pinson, 1990)), the use of ascendant sets combined with improved immediate selections allows us to perform ascendant set-like adjustments for *both* kinds of selections. Future work may include a domain splitting feature, in order to integrate a better support for Constraint Programming approaches, and an incremental feature.

Table 2. CPU comparison hard instances of Job-Shop Scheduling Problems.

Instance	$O(n \log n)$ Stable version	$O(n^2)$ Classical version
ORB1	100	90.3
ORB2	100	214.8
ORB3	100	90.3
ORB4	100	86.0
ORB5	100	250.0
ORB6	100	88.9
ORB8	100	93.8
ORB9	100	89.6
ORB10	100	106.7
MT1010	100	161.9
Average	100	127.2

APPENDIX. $C(K_i)$ COMPUTATION WITH A RED–BLACK TREE

At each step, we need to compute

$$C(K_i) = \max_{J' \subseteq K_i} \left\{ \min_{j \in J'} \alpha_j + \sum_{j \in J'} p_j \right\}$$

Let us define a red–black data structure to compute $C(K_i)$.

Let $\mathcal{T}$ be a red–black tree. In the tree, we denote for any node k associated with operation k:

- k_l, k_r, k_f, its left successor, its right successor and its predecessor in $\mathcal{T}$.
- $\mathcal{L}_k, \mathcal{R}_k$, its associated left and right subtrees (if $\mathcal{L}_k = \emptyset$ (resp. $\mathcal{R}_k = \emptyset$) then $k_r = 0$ (resp. $k_l = 0$)).
- $\mathcal{F}_k$, the subtree of root k.
- v, the root of $\mathcal{T}$.

$\mathcal{T}$ verifies the property

$$\forall k \leq n, \forall i \in \mathcal{L}_k, \forall j \in \mathcal{R}_k, \quad \alpha_i < \alpha_k \leq \alpha_j$$

Let us assign to any node k, the following quantities:

$$\sigma_k = p_k + \sum_{j \in \mathcal{R}_k} p_j \tag{A.1}$$

$$\zeta_k = \begin{cases} \max_{j \in \mathcal{F}_k} \left\{ \alpha_j + \sum_{i \in \mathcal{F}_k | \alpha_i \geq \alpha_j} p_i \right\} & \text{if } \mathcal{F}_k \neq \emptyset \\ -\infty & \text{otherwise} \end{cases} \tag{A.2}$$

From (A.1)–(A.2), we can deduce the following recursive definitions:

$$\sigma_k = p_k + \tau_{k_r} \tag{A.3}$$

$$\tau_k = p_k + \tau_{k_l} + \tau_{k_r} \tag{A.4}$$

$$\zeta_k = \max\{\sigma_k + \zeta_{k_l}\,;\ \alpha_k + \sigma_k\,;\ \zeta_{k_r}\} \tag{A.5}$$

with the convention that $\sigma_0 = 0$, $\tau_0 = 0$ and $\zeta_0 = 0$.
Indeed, we have

$$\zeta_{k_l} = \max_{j \in \mathcal{L}_k} \left\{ \alpha_j + \sum_{i \in \mathcal{L}_k | \alpha_i \geq \alpha_j} p_i \right\}$$

$$\zeta_{k_r} = \max_{j \in \mathcal{R}_k} \left\{ \alpha_j + \sum_{i \in \mathcal{R}_k | \alpha_i \geq \alpha_j} p_i \right\}$$

In particular, it is straightforward to check that

$$\zeta_v = \max_{j \in \mathcal{T}} \left\{ \alpha_j + \sum_{i \in \mathcal{O} | \alpha_i \geq \alpha_j} p_i \right\} = \max_{J' \subseteq K} \left\{ \min_{j \in J'} \alpha_j + \sum_{j \in J'} p_j \right\} = C(K) \tag{A.6}$$

In a red–black tree, the following property holds: "if a function f for a node x can be computed using only the information in nodes k, k_l and k_r, then we can maintain the values of f in all nodes of $\mathcal{T}$ during insertion and deletion without affecting the $O(\log n)$ performance of these operations." (See Cormen *et al.*, 1989.) The relations (A.3)–(A.5) verifying this property in the red–black tree $\mathcal{T}$, we can compute $C(K_i)$ in $O(\log n)$. The procedure **insert**(K, i) inserts operation i in the red–black tree K and returns the ζ-value of the root, $\zeta_v = C(K_i)$ according to relation (A.6).

References

Brinkkötter, W. and Brucker, P. (2001) Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. *Journal of Scheduling*, **4**:53–64.

Brucker, P., Jurisch, B. and Krämer, A. (1994) The job-shop problem and immediate selection. *Annals of Operations Research*, **50**:73–114.

Carlier, J. (1975) Thèse de 3e cycle, Paris VI.

Carlier, J. and Pinson, É. (1989) An algorithm for solving the job-shop problem. *Management Science*, **35**:165–176.

Carlier, J. and Pinson, É. (1990) A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, **26**:269–287.

Carlier, J. and Pinson, É. (1994) Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, **78**:146–161.

Cormen, T., Leiserson, C. and Rivest, R. (1989) Introduction to Algorithms. MIT Press, Cambridge, MA.

Martin, P. and Shmoys, D. B. (1996) A new approach to computing optimal schedules for the job-shop scheduling problem, in: *Proceedings of the 5th International IPCO Conference*, pp. 389–403.

Péridy, L. and Rivreau, D. (2005) Local adjustments: a general algorithm. *European Journal of Operational Research*, **164**:24–38.

Tarski, A. (1955) A lattice-theoretical fixpoint theorem and its applications, *Pacific Journal of Mathematics*, **5**:285–309.

AN EFFICIENT PROACTIVE–REACTIVE SCHEDULING APPROACH TO HEDGE AGAINST SHOP FLOOR DISTURBANCES

Mohamed Ali Aloulou* and Marie-Claude Portmann
MACSI team of INRIA-Lorraine and LORIA-UHP-INPL
Ecole des Mines de Nancy Parc de Saurupt,
54042 Nancy Cedex, France
aloulou@lamsade.dauphine.fr, portmann@loria.fr

Abstract We consider the single machine scheduling problem with dynamic job arrival and total weighted tardiness and makespan as objective functions. The machine is subject to disruptions related to late raw material arrival and machine breakdowns. We propose a proactive–reactive approach to deal with possible perturbations. In the proactive phase, instead of providing only one schedule to the decision maker, we present a set of predictive schedules. This set is characterised by a partial order of jobs and a type of associated schedules, here semi-active schedules. This allows us to dispose of some flexibility in job sequencing and flexibility in time that can be used on-line by the reactive algorithm to hedge against unforeseen disruptions. We conduct computational experiments that indicate that our approach outperforms a predictive reactive approach particularly for disruptions with low to medium amplitude.

Keywords: scheduling, single machine, flexibility, robustness, total weighted tardiness, genetic algorithms.

1. INTRODUCTION

Scheduling is an important element of production management systems because it allows improving system performance and serves as an overall plan on which many other shop activities are based. For numerous scheduling problems, many techniques have been proposed to generate a unique schedule that provides optimal or near-optimal performance. However, when this precomputed or predictive schedule is released for execution, continual adapting

*Since September 2003, Mohamed Ali Aloulou is a member of laboratory LAMSADE, Université Paris Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France.

is required to take into account the presence of uncertainties. These uncertainties are related, for example, to machine breakdowns, staffing problems, unexpected arrival of new orders, early or late arrival of raw material and uncertainties in processing times (Davenport and Beck, 2000). When the first perturbations arise, the schedule is slightly modified and the performance is a little bit affected. But when there are more important perturbations, the performance of the final schedule becomes generally much worse than the initial one. Besides, the predictive schedule is also used as a basis for planning activities, such as raw material procurement, preventive maintenance and delivery of orders to external or internal customers (Mehta and Uzsoy, 1999). Consequently, if the obtained schedule deviates considerably from the initial predictive schedule, then this may delay the execution of many activities related to internal customers. It may also add some costs due to early procurement of raw material from suppliers or late delivery of finished products to external customers. Therefore, we are interested in developing an approach that builds predictive schedules that provide a sufficiently detailed sketch of the schedule to serve as the basis for other planning requirements and retain enough flexibility in order to hedge against perturbations that may occur on-line and/or minimise their effects on planned activities.

Many approaches of scheduling and rescheduling have been proposed in the literature to take into account the presence of uncertainties in the shop floor. They can be classified into four categories: completely reactive scheduling, predictive reactive scheduling, proactive scheduling and proactive–reactive scheduling; see for example Davenport and Beck (2000) and Herroelen and Leus (2003).

"Completely reactive approaches are based on up-to-date information regarding the state of the system" (Davenport and Beck, 2000). No predictive schedule is given to the shop floor and the decisions are made locally in real time by using priority-dispatching rules. These approaches are so widespread in practice because they are easy to implement and use. They are particulary competitive when the level of disturbances is always important or when the data are known very late, making impossible the computation of predictive schedules. In predictive reactive approaches, a predictive schedule is generated without considering possible perturbations. Then, a reactive algorithm is used to maintain the feasibility of the schedule and/or improve its performances; see for example Church and Uzsoy (1992) and Vieira *et al.* (2003). The goal of proactive or robust scheduling is to take into account possible disruptions while constructing the original predictive schedule. This allows one to make the predictive schedule more robust. A robust schedule is defined by Leon *et al.* (1994) as a schedule that is insensitive to unforeseen shop floor disturbances given an assumed control policy. This control policy is generally simple. Clearly, robust scheduling is appropriate only if, while generating the

predictive schedule, the uncertainty is known or at least some suspicions about the future are given thanks to the experience of the decision maker.

"A scheduling system that is able to deal with uncertainty is very likely to employ both proactive and reactive scheduling" (Davenport and Beck, 2000). Indeed, it is very difficult to take into account all unexpected events while constructing the predictive schedule. Consequently, a reactive algorithm more elaborate than those used in proactive or robust approaches should be used to take into account these improbable events. However, due to the constraints on the response time of the reactive algorithm, one cannot expect an optimal or near-optimal decision. That is why it is interesting that the proactive algorithm provides solutions containing some built-in flexibility in order to minimise the need of complex search procedures for the reactive algorithm. Several approaches that more explicitly use both off-line (proactive) and on-line (reactive) scheduling have been developed in the literature: see for example Artigues *et al.* (1999), Billaut and Roubellat (1996) and Wu *et al.* (1999).

The approach we present here is a proactive–reactive approach. In the first step, we build a set of schedules restricted to follow a partial order of jobs, which allows us to introduce some flexibility in the obtained solution. Then, this flexibility is used on-line to hedge against some changes in the shop environment. It can also be used by the decision maker to take into account some preferences or non-modelled constraints.

In this paper, we consider the problem of scheduling a set $N = \{1, \ldots, n\}$ of jobs on a single machine. Each job $j \in N$ has a processing time $p_j > 0$, a release time $r_j \geq 0$, a due date $d_j \geq 0$ and a weight $w_j > 0$. The shop environment is subject to perturbations that can be modelled by the increase of the release dates of some jobs and by machine breakdowns. The objective functions considered here are the total weighted tardiness $\bar{T}_w = \sum_{j=1}^{n} w_j T_j$ and the makespan $C_{\max} = \max\{C_j\}$, where C_j and $T_j = \max\{0, C_j - d_j\}$ are, respectively, the job completion time and the tardiness of job $j, j = 1, \ldots, n$, of a given schedule. The preemption of job processing is allowed only if a machine breakdown occurs.

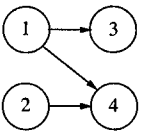
The rest of the paper is organised as follows. Sections 2 and 3 are dedicated, respectively, to the proactive algorithm and the reactive algorithm. Before concluding, our approach is compared, in Section 4, to a predictive–reactive approach in a supply chain context and in stochastic shop conditions.

2. THE PROACTIVE ALGORITHM

In this section, we define a solution to the single machine scheduling problem presented in the introduction and explain its interest. Then, we present different measures to evaluate a solution performance and flexibility. Finally,

Job j	r_j	p_j	d_j	w_j
1	0	3	6	1
2	1	3	8	1
3	0	2	9	1
4	9	2	12	1

(a)



(b)

Figure 1. A four-job problem: (a) a numerical example, (b) a partial order.

we present the genetic algorithm used to compute solutions providing a good trade off between the defined quality measures.

2.1 Definition of a Solution to the Problem

As noticed in the surveys of Davenport and Beck (2000) and Herroelen and Leus (2003) on scheduling with uncertainty, introducing flexibility in the solutions computed off-line allows for increasing the robustness of the system. Hence, we consider that a solution to the problem is not a particular schedule but a set of schedules characterised by a structure defined by a partial order of jobs and a type of schedules. The schedule type can be semi-active, active or non-delay, see for example Baker (1974). For a semi-active schedule, a job cannot be shifted to start earlier without changing the job sequence or violating the feasibility (here precedence constraints and release dates). An active schedule is a schedule where no job can be shifted to start earlier without increasing the completion time of another job or violating the feasibility. A schedule is called non-delay if the machine does not stand idle at any time when there is a job available at this time. Observe that the set of non-delay schedules is a subset of the set of active schedules which is in turn a subset of the set of semi-active schedules.

In Figure 1, we consider a four-job problem and a partial order where the only restrictions are that job 1 precedes jobs 3 and 4 and job 2 precedes 4. This partial order represents two non-delay schedules S_1 and S_2, three active schedules S_1, S_2, and S_3 and five semi-active schedules $S_1, \ldots, S_5$, see Figure 2. The corresponding objective function values are also given in the same figure.

It is clear that proposing several good schedules is more interesting than proposing only one. The decision maker can consequently choose the schedule that better responds to his/her preferences and possibly to non-modelled constraints. It is even more interesting if some proposed schedules have common characteristics allowing one to switch from one schedule to another easily. In this case, the decision maker can postpone the choice of the schedule to be

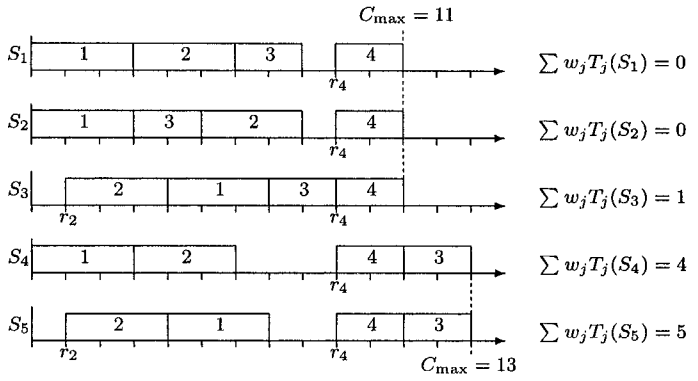


Figure 2. Represented non-delay, active and semi-active schedules.

executed. It is only on-line that he makes this decision, taking into account up-to-date information regarding the state of the system.

The structure defined by the couple (partial order of jobs, type of schedules) allows one to represent several schedules having common precedence constraints between jobs. In order to switch from one schedule to another, or from one subset of schedules to another subset, one or several edges in the disjunctive graph representing the partial order should be oriented without creating cycles in the obtained graph. Furthermore, a partial order is a natural structure used in iterative methods. In such approaches, one or several arcs are added in each solution step. The use of transitive graphs allows one to guarantee the feasibility of the obtained schedule.

2.2 Definition of the Quality of a Solution

The example discussed in Section 2.1 shows that the quality a priori of a partial order depends on the type of schedules considered. Indeed, for semi-active schedules, there are fewer restrictions and this allows one to obtain five represented schedules instead of two and three for, respectively, non-delay and active schedules. The number of represented schedules can measure the flexibility in job sequencing of a solution. Considering semi-active schedules allows one to improve the flexibility of a solution according to the proposed measure. However, the worst performance of the represented schedules is worse than when considering active or non-delay schedules (see the performance of schedules S_4 and S_5).

In our first implementation, we consider that the schedules are restricted to be semi-active. According to this choice, we define in the following the different quality measures of a solution.

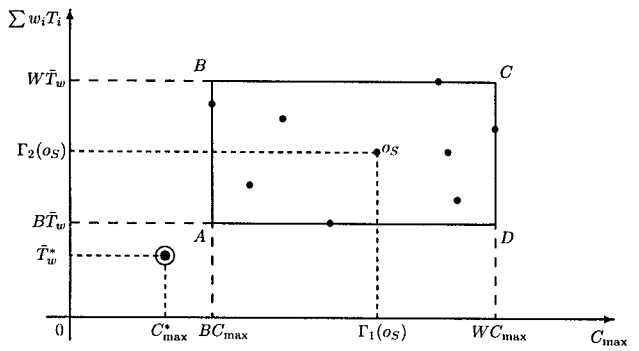


Figure 3. A solution S in $(C_{\max}, \bar{T}_w)$ space.

2.2.1 Performance of a Solution.

2.2.1.1 Definition of the performance. Let S be a solution to the problem. It is a partial order that represents a set of semi-active schedules o_S. To each schedule $o_S \in S$ is associated a vector $\Gamma(o_S) = (\Gamma_1(o_S), \Gamma_2(o_S))$, where $\Gamma_1(o_S)$ is the makespan of o_S and $\Gamma_2(o_S)$ its total weighted tardiness. Hence, a solution S is represented by the set of vectors $\Gamma(o_S)$. Figure 3 represents the vectors $\Gamma(o_S)$ in the $(C_{\max}, \bar{T}_w)$ space.

The performance of solution S is closely related to the objective function values of all represented schedules o_S. Since the number of these schedules can be rather large, it is natural to consider only their most important representatives. We suggest using only the best- and the worst-case performances to evaluate the performance of a flexible schedule. The best-case performance provides a lower bound of the performance when following the partial order, whereas the worst-case performance gives a guarantee regarding how poorly the schedules may perform when following the partial order. They can be obtained by solving corresponding minimisation and maximisation problems. Hence, we have to determine the coordinates of points A, B, C, and D shown in Figure 3. The rectangle formed by these points represents the range of the objective function values of all the schedules following the considered solution.

We denote by *goal point* the point in $(C_{\max}, \bar{T}_w)$ space whose coordinates are respectively the best makespan $C^*_{\max}$ and the best total weighted tardiness $\bar{T}^*_w$ of the problem (without additional precedence constraints brought by the partial order given by S), see Figure 3.

A solution S is all the more efficient that the distance between the goal point to each point A, B, C, and D is small. Hence, a performance measure of a solution S can be defined as a linear combination D_S of the distances between

the goal point and the four points A, B, C, and D (see Figure 3). These points are determined by computing the best and worst makespan, denoted by $BC_{\max}$ and $WC_{\max}$, and best and worst total weighted tardiness, denoted by $B\bar{T}_w$ and $W\bar{T}_w$. We have

$$D_S = \alpha D_S^1 + (1-\alpha) D_S^2 \quad (1)$$

$$D_S^1 = \beta \frac{BC_{\max} - C^*_{\max}}{C^*_{\max}} + (1-\beta)\frac{WC_{\max} - C^*_{\max}}{C^*_{\max}}, \beta \in [0,1] \quad (2)$$

$$D_S^2 = \gamma \frac{B\bar{T}_w - \bar{T}^*_w}{\bar{T}^*_w + 1} + (1-\gamma)\frac{W\bar{T}_w - \bar{T}^*_w}{\bar{T}^*_w + 1}, \gamma \in [0,1] \quad (3)$$

The lower is D_S, the higher the performance of solution S is.

2.2.1.2 Computation of the performance. The objective functions we consider are the total weighted tardiness $\bar{T}_w$ and the makespan $C_{\max}$. It is proved that the problem of minimising the makespan w.r.t. precedence constraints, denoted by $1|prec, r_j|C_{\max}$, can be solved in $O(n^2)$ time (Lawler, 1973). The problem of minimising the total weighted tardiness, denoted by $1|prec, r_j|\sum w_j T_j$ is NP-hard in the strong sense, see for example Lawler *et al.* (1993). Consequently, for this latter problem we implemented a genetic-algorithm-based heuristic and made comparison with known dynamic dispatching rules like ATC, X-RM, and KZRM (Morton and Pentico, 1993). The results showed that the genetic algorithm we implemented is efficient but is time consuming for a large number of jobs, see Aloulou and Portmann (2001).

In order to compute the worst makespan and the worst total weighted tardiness that can be reached when considering a partial order S, we introduced in Aloulou *et al.* (2004) new optimisation problems. These are maximisation problems denoted by $1(sa)|prec, r_j|(F \rightarrow \max)$, where F is the objective function to be maximised, and notation sa means that semi-active schedules are considered.

We proved that the problem $1(sa)|prec, r_j|(C_{\max} \rightarrow \max)$ can be solved in $O(n^2)$ times and the problem $1(sa)|prec, r_j|(\sum w_j T_j \rightarrow \max)$, is NP-hard in the strong sense (Aloulou *et al.*, 2004). We developed several heuristics based on genetic algorithms and dispatching rules. We obtained the same conclusions as for minimisation problems, see Aloulou and Portmann (2001).

As a result, for a given solution (a set of schedules w.r.t. a partial order), we use polynomial-time algorithms to compute the exact minimal and maximal makespan and approximate algorithms based on dispatching rules for estimated minimal and maximal total weighted tardiness.

2.2.2 Flexibility of a Solution. One of the main characteristics of a solution to the problem is its flexibility. According to the example presented in the previous section, a solution is all the more flexible if the number of represented schedules is great. Indeed, when we have many schedules, it would be possible to dispose on-line, every time a decision has to be taken, of more than one alternative. This could allow one to hedge against some perturbations such as raw material availability. This flexibility, called *flexibility in job sequencing*, can be measured by the number of different schedules feasible w.r.t. the partial order. However, the problem of calculating this number is $\#P$-complete (Brightwell and Winkler, 1991). Thus, we propose another measure $Flex_{seq}$ equal to the number of non-oriented edges in the transitive graph representing the partial order. If this number is great, then the associated solution is flexible. Notice that the number of non-oriented edges in a transitive graph is inversely proportional to the number of arcs in the same graph. In our proactive algorithm, we use a transformation of the number of arcs into a qualitative scale defining several flexibility levels. Each level is characterised by an interval $[nbArcsMin, nbArcsMax]$ giving the minimal and maximal number of arcs that a solution contains.

Furthermore, another type of flexibility can be provided according to the time window in which the jobs are executed. This flexibility is called *flexibility in time*. A measure of this flexibility $Flex_{time}$ can be given by the ratio between the time window in which the jobs can be executed and the total processing time of the jobs:

$$Flex_{time} = \frac{WC_{\max} - P}{P} \tag{4}$$

where $WC_{\max}$ is the worst makespan of the considered solution and P is the total processing time of all the jobs.

2.3 A Genetic Algorithm to Achieve Scheduling Flexibility

In this section, we present a genetic algorithm whose goal is to build-up off-line the set of Pareto solutions conciliating good shop performance and flexibility presence (see T'kindt and Billaut (2002) for the definition of Pareto optimal solution). These solutions are determined by an exploration of the whole solution space. In practice, this exploration may be limited to a subspace of solutions satisfying possible preferences of the decision maker. These preferences are accounted for thanks to an interactive support decision tool, which is not presented in this paper.

In the following, we present the general scheme of the genetic algorithm we implemented. Then, we describe the selection and reproduction strategies, the

encoding used to represent a solution to the problem and the genetic operators that generate new solutions.

2.3.1 General Scheme of the Genetic Algorithm. The aim of this algorithm is to generate for a given problem a set of solutions that yields a good compromise between flexibility and performance. According to the previous section, a solution S is judged better when the distance $D_s(S)$ is minimal, the flexibility in job sequencing $flex_{seq}(S)$ is maximal, and the flexibility in time $flex_{time}(S)$ is maximal.

The principle of the algorithm is to work on different populations of solutions that belong to a same level of flexibility in job sequencing. Recall that a level of flexibility is defined by an interval $[nbArcsMin, nbArcsMax]$ giving the minimal and maximal number of arcs that a solution partial order can contain. The objective is to find, in the space of solutions that belong to the same flexibility in job sequencing level, the solutions realising a good compromise between the performance (measured by D_s) and the flexibility in time (measured by $Flex_{time}$). We use a linear combination of these two criteria to compute the fitness of a chromosome representing a solution S:

$$Fitness(S) = \theta D_s(S) - (1-\theta)Flex_{time}(S), \text{where } \theta \in [0,1] \quad (5)$$

For a fixed parameter θ, the algorithm works as described in Figure 4. In this algorithm, we use the following notation:

- $nbGen$ is the number of generations;
- P_i is the population of individuals in iteration $i = 1, \ldots, nbGen$;
- $nbLevels$ is the number of levels of flexibility in job sequencing;
- $Elite_{L^l,\theta}$, is the elite population obtained at the end of the algorithm for a fixed value $\theta \in [0,1]$ and for the level of flexibility in job sequencing $L^l, l = 1, \ldots, nbLevels$.

In order to find the Pareto optimal solution set of this multi-criteria problem, we vary the value of θ between 0 and 1 and apply the previous algorithm for each value of θ.

2.3.2 Selection and Reproduction Strategy. We use two selection procedures in each iteration of the genetic algorithm. The first selects N_{pop} couples of chromosomes for reproduction. The second selection procedure selects, among the generated children and the old population, N_{pop} chromosomes that will survive and form the new population. Our implementation uses roulette selection by rank for the two previous procedures. Denote by N_{popIn} the number of chromosomes of the initial population from which the

For $l = 1$ to $nbLevels$ do
  Generate a population of solutions P_0 belonging to a level of flexibility in job sequencing L^l;
  Evaluate the chromosomes in P_0 and initialise $Elite_{L^l,\theta}$;
  For $i = 0$ to $nbGen$ do
    Select, from P_i, N_{pop} couples of chromosomes for reproduction;
    Cross the selected couples of chromosomes;
    Evaluate the generated children and update $Elite_{L^l,\theta}$;
    Mutate with a small probability π_{mut} the generated children;
    Evaluate the mutated children and update $Elite_{L^l,\theta}$;
    Select, from P_i and the generated children, N_{pop} chromosomes for survival;
  EndFor
EndFor

Figure 4. The genetic algorithm for a fixed value of θ.

procedure selects N_{popOut} chromosomes for reproduction or survival. The best chromosome is assigned a new fitness equal to N_{popIn}, the second best chromosome a fitness equal to $N_{popIn} - 1$ and the last one a fitness equal to 1. The chromosomes are selected with a chance proportional to their rank.

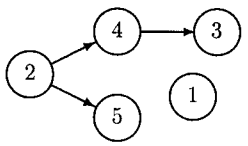
2.3.3 Encoding. To encode a given solution S (partial order), we use a ternary precedence constraint-oriented matrix A, which represents the set of precedence constraints contained in the transitive graph representing the partial order. This matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ is defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if job } i \text{ precedes job } j, \\ -1 & \text{if job } j \text{ precedes job } i, \\ 0 & \text{if } i = j \text{ or } i \text{ and } j \text{ are permutable} \end{cases}$$

This matrix is transitive and anti-symmetric. Only $n(n-1)/2$ elements are kept inside the computer memory, but the complete matrix is used here for clarity sake.

It is a direct encoding because there is a one-to-one correspondence between the ternary matrix space and the solution space (Djerid and Portmann, 1996; Portmann *et al.*, 1998).

Example 1 Consider a five-job problem. A solution to this problem is given by the partial order represented in Figure 5. In this solution, the only restrictions are that job 2 precedes jobs 3, 4, and 5; and job 4 precedes job 3. This solution represents 15 schedules. It is encoded by the ternary matrix given in Figure 5.



Jobs	1	2	3	4	5
1	0	0	0	0	0
2	0	0	1	1	1
3	0	−1	0	−1	0
4	0	−1	1	0	0
5	0	−1	0	0	0

Figure 5. A solution and the corresponding encoding.

Step 1. Compute $F = \frac{A^1+A^2}{2}$;*//integer division: the value is truncated to* 0 *when obtaining* $\frac{1}{2}$ *or* $-\frac{1}{2}$
Update $nbArcs(F)$;

Step 2. Make a decision on the value of one f_{ij} to ensure, if possible, that the child will be different from both mates;
Update $nbArcs(F)$;

Step 3. While $nbArcs(F) < nbArcs(A^1)$ do
Select randomly mate 1 with probability π or mate 2 with probability $1 - \pi$;
Select randomly two jobs i and j such that $f_{ij} = 0$ and $a_{ij} \neq 0$;*//A is the matrix of the selected mate*
set $f_{ij} = a_{ij}$;
set $f_{ji} = -f_{ij}$;
Compute the transitive closure of F and update $nbArcs(F)$;
EndWhile

Step 4. if $nbArcs(F) \leq nbArcsMax$ then F is kept;
else F is discarded;

Figure 6. The modified MT3 crossover.

2.3.4 Modified MT3 crossover. In order to generate two children from two mates, we use an adaptation of the crossover MT3 proposed by Djerid and Portmann (1996), which was used for the job shop scheduling problem. Consider two mates (mate 1 and mate 2) represented respectively by matrix A^1 and matrix A^2. The algorithm corresponding to the crossover that generates the first child, represented by matrix F, works as described in Figure 6.

Step 1 allows us to keep the common precedence constraints in the two parents. In step 3, we add 1 and -1 (precedence constraints) in the matrix of the child according to the parents until reaching $nbArcs(A^1)$ (or $nbArcs(A^2)$). The probability π, which we choose equal to 0.6 in the implementation of the algorithm, allows that child 1 (resp. child 2) resembles more to mate 1

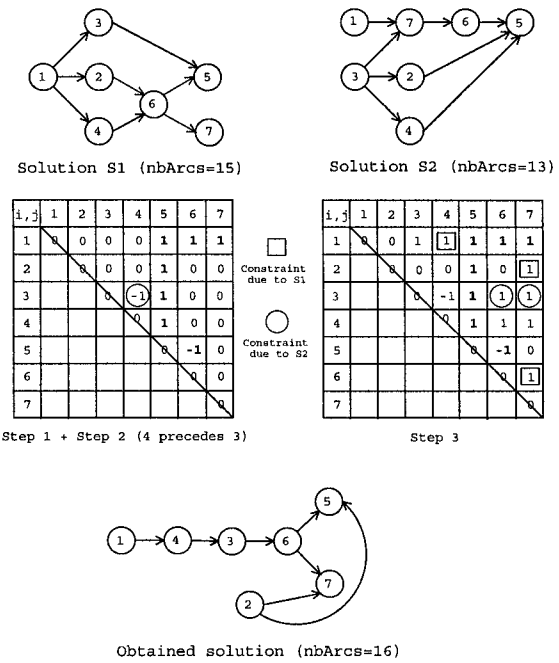


Figure 7. MT3 crossover example.

(resp. mate 2). In Figure 7, we present an example that illustrates the proposed crossover.

Remarks

- The crossover presented below guarantees an interesting property: *if i precedes j in both mate 1 and mate 2, then i precedes j in the generated offspring*. This allows the maintenance of important properties of a good solution. Furthermore, it guarantees that imperative precedence constraints, if they exist, are usually respected.

- If the obtained offspring F is such that $nbArcs(F) > nbArcsMax$, then we can withdraw the obtained solution and reiterate or try to eliminate some arcs in order to obtain $nbArcs(F)$ almost equal to $nbArcsMax$.

- Unlike the crossover proposed by Djerid and Portmann (1996), the presented crossover stops adding 1 and -1 when the number of arcs in the generated offspring reaches the number of arcs of the corresponding

Step 1. Initialise $F = F_0$; /* F_0 *is the matrix representing the imposed precedence constraints* */

Step 2. Select randomly two jobs i and j such that $a_{ij} \neq 0$;
set $f_{ij} = -a_{ij}$;
set $f_{ji} = -f_{ij}$;
While $nbArc(F) < nbArc(A)$
Select randomly two jobs i and j such that $f_{ij} = 0$ and $a_{ij} \neq 0$;
set $f_{ij} = a_{ij}$;
set $f_{ji} = -f_{ij}$;
Compute the transitive closure of F;
EndWhile

Step 3. if $nbArcs(F) \leq nbArcsMax$ then F is kept ;
else F is discarded ;

Figure 8. The mutation MUT3.

mate. Besides, step 2 is used to ensure that the generated child is different from his parents. This step is motivated by the chosen scheme of the genetic algorithm: Davis reproduction (Davis, 1991).

2.3.5 MUT3 Mutation. The mutation operator is used to guarantee the diversity of the population of chromosomes. The mutation we propose consists in changing the order of at least two jobs. It is described in Figure 8, where A is the matrix of the considered mate.

3. THE REACTIVE ALGORITHM

The reactive algorithm has three important roles. The first role is to make the remaining scheduling decisions on-line w.r.t the precedence constraints imposed by the chosen partial order. The second role is to react when a perturbation occurs. The third role is to detect when the solution in use is infeasible w.r.t. the decision maker objectives.

A reactive algorithm is efficient if it

1 offers, if possible, every time a decision has to be made, more than one alternative w.r.t. the partial order and consequently be able to absorb possible late arrival of raw material,

2 exploits the flexibility in time when a machine breaks down, and

3 obtains good performance for the realised schedule.

Set $X = N$ and $t = 0$;
While $X \neq \phi$ do
  Determine X^+;
  Determine the set $Y_1 = \{i \in X^+, PRIOR_1(i,t) = \max_{j \in X^+}\{PRIOR_1(j,t)\}\}$;
  Select a job $i^* \in Y_1$ such that $PRIOR_2(i^*,t) = \max_{j \in Y_1}\{PRIOR_2(j,t)\}\}$;
  Set $X = X \backslash \{i^*\}$;
  Set $t = max\{t, r_{i^*}\} + p_{i^*}$;
endWhile

Figure 9. The general scheme of the proposed algorithms.

Clearly, it is impossible to satisfy these objectives simultaneously. Hence, we designed different procedures to achieve an acceptable compromise between the aforementioned objectives. The following notation is used in the remainder of the paper:

- For a set of jobs X, X^+ is the subset of available jobs. An unscheduled job is available if all its predecessors (w.r.t. the partial order) are scheduled and if it satisfies the restrictions on the constructed schedules (semi-active, active or non-delay schedules).
- $PRIOR_1(i,t)$ and $PRIOR_2(i,t)$ are two functions that give the priority of a job i at time t.

The general scheme of the reactive algorithms we propose, in absence of disruptions, is given in Figure 9. At a given time t, the algorithms schedule the job i^* that maximises $PRIOR_1(i,t)$ among the available jobs i, i.e. $i \in X^+$. When two or more jobs are in competition, the job that maximises $PRIOR_2(i,t)$ is selected.

Clearly, the algorithms depend on the priority functions $PRIOR_1(i,t)$ and $PRIOR_2(i,t)$. They also depend on the definition of the job availability. Indeed, even though the partial order was computed off-line while considering that the represented schedules are semi-active, we may decide on-line to construct active or non-delay schedules for a performance improving purpose.

When a disruption occurs, an analysis module is used. In the case of a breakdown, this module uses some results developed in Aloulou (2002) to compute an upper bound of the increase on the worst total weighted tardiness and to propose, if possible, one or several decisions to minimise it. The increase on the worst makespan can be computed with a polynomial time algorithm (Aloulou *et al.*, 2004). We consider now the second type of disruptions: late job arrival.

Suppose that, at time t, the algorithm chooses to schedule a disrupted job j such that $r_j^m > t$, where r_j^m is the new release date of the job. If there exists an available non-disrupted job i, it can be scheduled without any increase on the worst-case objective function's values. If such a job does not exist, then this perturbation can be considered as a breakdown and an upper bound of the increase on the worst total weighted tardiness can be computed. When the loss of performance is small (under a given threshold), we schedule the job that minimises the loss. When the loss of performance is too important, we can either restrain the partial order in order to have an acceptable increase or re-calculate a new solution (partial order) for the remaining non-scheduled jobs (re-scheduling).

4. COMPUTATIONAL RESULTS

We conducted several experiments to evaluate the proactive algorithm and the use of the proposed approach both in deterministic and stochastic shop conditions. The first type of experiment concerns the proactive algorithm and its ability to construct solutions providing a good trade-off between the measures of flexibility and performance. The experiments showed that every solution S, computed by the proactive algorithm, has a best-case performance $B\bar{T}_w(S) \leq 1.04 \times \bar{T}_w^*$. This implies that S contains at least one schedule that has a comparable performance to the best schedule given by the heuristics ATC, X-RM and KZRM. Further, in some cases, the best performance is even improved by 20%. We also noticed that the genetic algorithm allows one to provide an acceptable trade-off between flexibility and performance when the dispersion of job arrival is not very high. Otherwise, the job permutation must be limited to a low flexibility level in order to preserve correct performance (Aloulou, 2002).

In the second type of experiments, we tested the application of the approach in deterministic shop conditions. We showed that the best two algorithms in deterministic shop conditions are $Perf\_ND$ and $Flex1\_ND$. $Perf\_ND$ constructs non-delay (ND) schedules using ATC cost as the first priority function ($PRIOR_1(i,t)$) and the non-decreasing order of release dates as the second priority function ($PRIOR_2(i,t)$). $Flex1\_ND$ tries to maximise the use of the flexibility in job sequencing contained in the considered solution by maximising the number of possible choices in the following step. The second priority rule is ATC (Aloulou, 2002). In this paper, only experiments in stochastic shop conditions case are detailed.

4.1 Experimentation in Stochastic Shop Conditions

In order to evaluate the proposed proactive–reactive approach, we compare it to a predictive reactive approach. In the proactive–reactive approach, the re-

active algorithm, presented in the previous section, allows only small modifications on the partial order characterising the retained solution. In the predictive reactive approach, a predictive schedule is computed using KZRM heuristic without taking into account the presence of future disruptions. This schedule is then proposed to the shop for execution. When a disruption occurs, a reactive algorithm, based on ATC heuristic, is used to control the shop until the next rescheduling. For both approaches, the rescheduling is assumed to be made after the period concerned by the experimentations. In our implementation, we try not to favour, or handicap, either of these two approaches.

We show in the following sections that our proactive–reactive approach has a better behaviour then the predictive reactive approach when disruptions have low or medium amplitude.

4.1.1 Description. The experimentation's context is inspired from problems we met in a supply chain environment (*Growth project V-chain GRD1-2000-25881*). In this environment, in order to process final products, several components are needed: principal components and secondary components.

- Principal components are delivered by an upstream shop. The release dates r_j of the jobs, used by the proactive and predictive algorithms, are equal to the arrival dates of the corresponding components.
- Secondary components are bought and used for the transformation of the principal components. We suppose that two different principal components do not need the same secondary components.

After acquisition, secondary components are stored in the shop until their use. We associate with each secondary component a storage cost proportional to its storage duration. Obtained products are then delivered to a downstream shop. A penalty is associated with each job if it is tardy w.r.t. the due dates d_j given by the downstream shop.

Two solutions S^{pro} and S^{pred} are computed off-line using, respectively, the proactive algorithm and the predictive algorithm. The computation of these solutions takes into account the delivery time of principal components r_j and the due dates d_j. In order to minimise the cost of secondary component storage, the purchase date of these components is a function of the starting time of the corresponding principal products in S^{pro} or S^{pred}. The finishing time of the principal components can determine their new delivery dates that can be communicated to the downstream shop. A second penalty is associated with each job if it is tardy w.r.t. the new delivery dates.

The following notation is used (see Figure 10). For each job j, we denote by

- p_j its processing time;

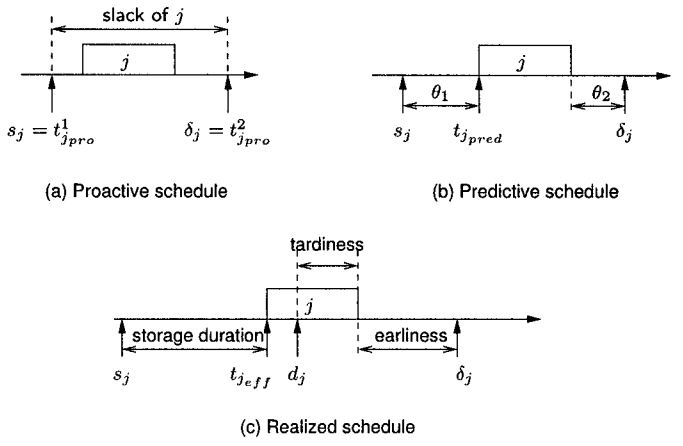


Figure 10. The different parameters for a job j in the proactive solution, the predictive schedule and the realised schedule.

- $t_{j_{pred}}$ its starting time in S^{pred};
- $t^1_{j_{pro}}$ its earliest starting time w.r.t. the partial order defining S^{pro};
- $t^2_{j_{pro}}$ its latest starting time w.r.t. the partial order defining S^{pro};
- $t_{j_{eff}}$ its effective starting time in the realised schedule S_r;
- s_j the secondary component acquisition date used to produce j:

$$s_j = \begin{cases} t_{j_{pred}} - \theta_1 & \text{for the predictive reactive approach} \\ t^1_{j_{pro}} & \text{for the proactive–reactive approach} \end{cases} \quad (6)$$

  where θ_1 is a constant (see Figure 10);
- d_j its due date asked by the downstream shop;
- δ_j its new delivery date given by the following:

$$\delta_j = \begin{cases} t_{j_{pred}} + p_j + \theta_2 = \delta^{pred}_j & \text{for the predictive reactive approach} \\ t^2_{j_{pro}} + p_j = \delta^{pro}_j & \text{for the proactive–reactive approach} \end{cases} \quad (7)$$

  where θ_2 is a constant (see Figure 10);
- T_j (resp. T^{δ}_j) its tardiness w.r.t. d_j (resp. δ_j).

The performance of the realised schedule S_r is function of three measures: the storage cost $storageCost(S_r)$, the tardiness $WT(S_r)$ w.r.t. the due dates d_j and the tardiness $WT^{\delta}(S_r)$ w.r.t. the delivery dates δ_j. These measures are given by the following equations:

$$storageCost(S_r) = \sum_{j \in N} c(t_{j_{eff}} - s_j) \tag{8}$$

where c is the unitary storage cost;

$$WT(S_r) = \sum_{j \in N} w_j T_j = \sum_{j \in N} w_j \max(0, t_{j_{eff}} + p_j - d_j) \tag{9}$$

$$WT^{\delta}(S_r) = \sum_{j \in N} w_j T_j^{\delta} = \sum_{j \in N} w_j \max(0, t_{j_{eff}} + p_j - \delta_j) \tag{10}$$

To be fair to the predictive approach, constants θ_1 and θ_2 are chosen in such a way that the predictive schedule S^{pred} is given a flexibility equivalent to the flexibility contained in the proactive solution S^{pro}. Constant θ_1 is such that the obtained storage costs are equivalent when considering S^{pro} or S^{pred}. The value of θ_2 is such that

$$\sum_{j \in N} (\delta_j^{pro} - d_j) \approx \sum_{j \in N} (\delta_j^{pred} - d_j) \tag{11}$$

4.1.2 Problem's Generation Scheme. The generation scheme is the one used by Mehta and Uzsoy (1999). The job's number is equal to $n = 40$. The processing times p_j are generated from a discrete uniform distribution $Uniform(p_{\min}, p_{\max})$. The job release times r_j are generated by a discrete uniform distribution between 0 and $\rho n p_{av}$, where p_{av} is the expected processing time. The parameter ρ controls the rate of job arrivals. The job due dates d_j are generated as $d_j = r_j + \gamma p_{av}$, γ is generated from a continuous distribution $Uniform[a, b]$. The job weights w_j are either all equal to 1 or generated by a discrete uniform distribution between 1 and 10.

4.1.3 Disruption's Generation Scheme. Two types of disruptions are considered: machine breakdowns and late arrival of principal components (increase of some r_j). The breakdowns are characterised by their occurrence time and their duration. The number of breakdowns is denoted by $nbBreak$. The scheduling horizon is divided into equal $nbBreak$ intervals such that in each interval a breakdown occurs. The breakdown durations are generated from a uniform distribution $Uniform(durMin, durMax)$.

The disruptions related to late arrival of the principal components are generated according to the considered predictive schedule or proactive solution.

When using a predictive schedule S^{pred}, a perturbation is generated by associating a new earliest starting time $r_j^m = t_{j_{pred}} + aug$, where $t_{j_{pred}}$ is the starting time of job j in S^{pred} and aug is generated from a uniform distribution $Uniform(augMin, augMax)$. When considering a proactive solution S^{pro}, $r_j^m = t^1_{j_{pro}} + aug$, where $t^1_{j_{pro}}$ is the earliest starting time of job j w.r.t. the partial order defining S^{pro}. For each problem, we consider a proactive solution and a predictive schedule to be compared. 50% of disruptions related to late raw material arrival are generated according to the predictive schedule and 50% according to the proactive solution. The number of delayed jobs is denoted $nbDelay$.

4.2 Analysis of the Obtained Results

We implement two heuristics of the reactive algorithm in the predictive reactive approach: ATC_d_j and ATC_δ_j. These are ATC-based heuristics that construct non-delay schedules. The first takes into account the due dates d_j and the second the delivery dates δ_j. For the proactive–reactive approach, we use the heuristics presented in Section 3: $Perf\_ND$ and $Flex1\_ND$. The comparison is based on the criteria WT and WT^δ given respectively by (9) and (10). The performance is measured by evaluating $WT + WT^\delta$.

We consider the following parameters to generate problems and disruptions:

- $(p_{\min}, p_{\max}) = (1, 11)$,
- $\rho = 0.5, 1$: grouped or dispersed job arrivals,
- $(a, b) = (1, 3)$ or $(2, 5)$ to generate the due dates.

For the four parameter combinations, we generate five problems. For each problem, we use the KZRM heuristic to generate a predictive schedule and the proactive algorithm to compute a flexible solution belonging to the flexibility level characterised by $nbArcs \in [620, 700]$. Recall that the number of jobs is equal to $n = 40$ and the maximum number of arcs is 780 $(= n(n-1)/2)$. This means that the proactive algorithm search space is formed by job partial orders where the proportion of non oriented arcs is between 10% and 20% of the total number of arcs. The predictive schedule and the proactive flexible solution are executed subject to disruptions characterised by the following parameters:

- the number of breakdowns $nbBreak \in \{0, 1, 2, 3\}$;
- $[durMin, durMax] = [3, 6]$ for breakdown durations;
- $[augMin, augMax] \in \{[1, 6], [6, 12], [12, 24], [24, 36], [36, 48], [60, 90]\}$;
- $nbDelay \in \{4, 6, 8\}$.

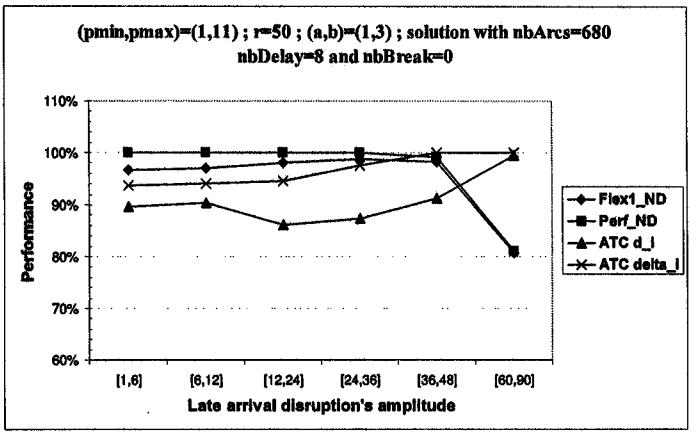


Figure 11. Results for $nbBreak = 0$.

For each combination of these parameters, a couple (predictive schedule, proactive solution) is experimented in 1000 different scenarios. The performance of each reactive algorithm $Flex1\_ND$, $Perf\_ND$, ATC_d_i and ATC_δ_i in terms of total weighted tardiness ($WT + WT^{\delta}$) is evaluated for each scenario. The average value is then computed and associated with each algorithm. The reactive algorithm with best performance is assigned a performance equal to 100%. Any of the three remaining algorithms is assigned the ratio between the best performance and its performance.

Figures 11–14 sum up the obtained results for the family of problems corresponding to $\rho = 0.50$, $(a, b) = (1, 3)$, $nbDelay = 8$ and $nbBreak = 0, 1, 2, 3$ and $nbArcs = 680$ for the proactive solution.

We can notice that algorithms ATC_d_j and ATC_δ_j are dominated by algorithms $Flex1\_ND$ and $Perf\_ND$ for late arrival disruptions characterised by $[augMin, augMax] \in \{[1, 6], [6, 12], [12, 24], [24, 36]\}$. The superiority of $Flex1\_ND$ and $Perf\_ND$ is very clear when $nbBreak = 2$. For important late arrival disruptions, notably when $[augMin, augMax] = [60, 90]$, the performance of $Flex1\_ND$ and $Perf\_ND$ become less than 86%. Besides, in all cases $Perf\_ND$ outperforms $Flex1\_ND$. This is also the case for ATC_δ_j when compared to ATC_d_j. We obtain the same conclusions when $(a, b) = (2, 5)$.

We applied the same experimentations for more flexible solutions with $nbArcs \approx 540$. We remarked that algorithms $Flex1\_ND$ and $Perf\_ND$ become rapidly dominated by algorithms ATC_δ_j and ATC_d_j. This means that asking for more flexibility is in this case penalising w.r.t. to the realised performance.

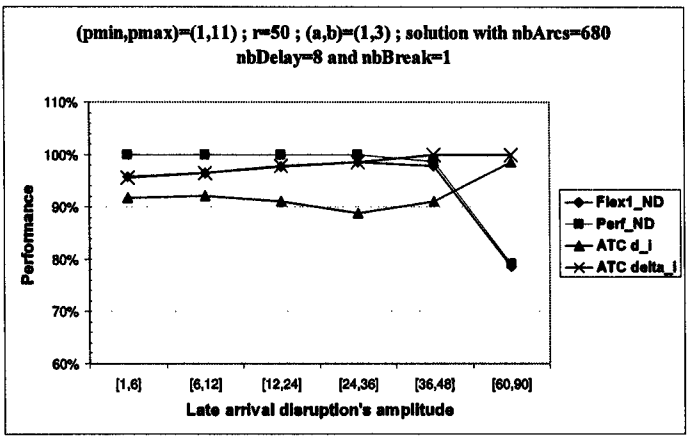


Figure 12. Results for $nbBreak = 1$.

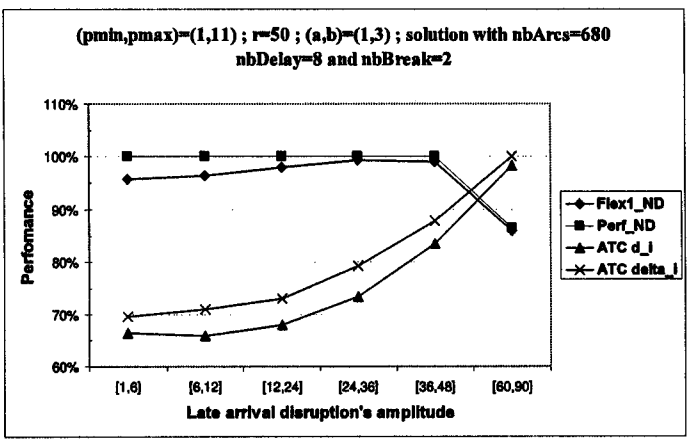


Figure 13. Results for $nbBreak = 2$.

When the dispersion of job arrival is high ($\rho = 1.00$), our algorithms $Flex1\_ND$ and $Perf\_ND$ outperform the other algorithms when the level of disruptions on job arrival is low ($[augMin, augMax] \leq [12, 24]$) and $nbBreak = 0$ or 1. When the number of breakdowns is greater, notably for $nbBreak = 2$, the performance of ATC_δ_j and ATC_d_j are less than 50%. Besides, in all cases $Perf\_ND$ outperforms $Flex1\_ND$. This is also the case for ATC_δ_j when compared to ATC_d_j.

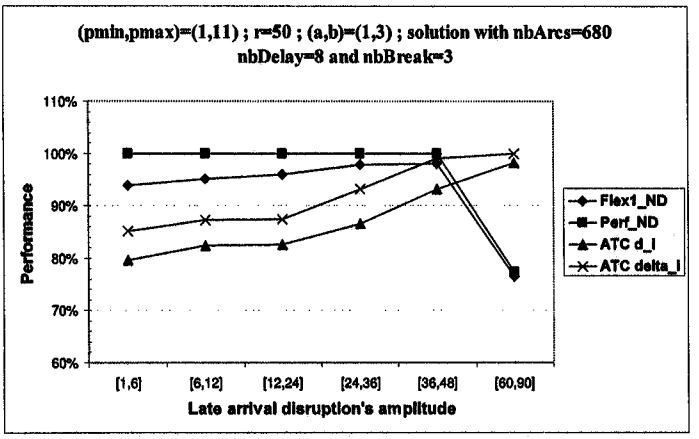


Figure 14. Results for $nbBreak = 3$.

To summarise, when the job arrivals are grouped ($\rho = 0.50$), our approach is usually superior to the predictive reactive approach when the amplitudes of disruptions on job arrival are such that $[augMin, augMax] \leq [36, 48]$ and $nbBreak = 0, 1, 2, 3$. For dispersed job arrivals ($\rho = 1.00$), our approach is very superior when $nbBreak \geq 2$. When $nbBreak < 2$, it also outperforms the other approach for disruptions on job arrival with low amplitude.

In conclusion, according to the experiments presented in this paper, our proactive–reactive approach outperforms the predictive reactive approach for low and medium disruption amplitude, especially when the flexibility level of the computed solutions is not too high (The number of non-oriented arcs is about 15% of the maximum number of arcs). Notice that the reactive algorithm of the predictive reactive approach can be considered as a rescheduling process since it uses the ATC heuristic, which is known to give good solutions w.r.t. total weighted tardiness criteria (Jouglet, 2002). Oppositely, the reactive algorithm used in our proactive approach is simple and allows only small modifications on the retained solution. However, in the presence of high-amplitude disruptions, such modifications may appear insufficient. Indeed, the proactive solution may become very bad and rescheduling is then necessary to regain some performance. That's why the predictive reactive approach seems to dominate the proactive–reactive one for very high disruption amplitude.

5. CONCLUSION

We have considered in this paper the single machine scheduling problem with dynamic job arrival and total weighted tardiness and makespan as objec-

tive functions. The shop environment is subject to perturbations related to late raw material arrival and to machine breakdowns. We proposed a proactive–reactive approach to solve the problem. The proactive algorithm is a genetic algorithm which computes partial orders providing a good trade-off between flexibility and performance. These solutions are built up while taking into account the shop constraints, the decision maker preferences and some knowledge about possible future perturbations. The reactive algorithm is used to guide on-line the execution of the jobs. In the presence of disruptions, the reactive algorithm is used to absorb their effects by exploiting the flexibility introduced in the proactive algorithm. We made several experimentations to compare our approach to a predictive reactive approach in stochastic shop conditions. In this comparison, we tried to be fair to the predictive reactive approach. We gave an equivalent flexibility range, on average, to the predictive schedule compared to the range of flexibility contained in a proactive solution. The results showed that our approach outperform the predictive reactive approach for low and medium disruption amplitude. This confirms that anticipating the presence of disruptions proactively allows to obtain good realised schedules and to plan other shop activities.

The results obtained for single machine scheduling problems gave us some insight to extend our approach to more complex scheduling problems. Currently, we are adapting the proposed proactive algorithm for flow shop scheduling problem with maximum cost functions. A solution to this problem is characterised by a partial order on each machine. This extension is motivated by a polynomial time algorithm developed in Aloulou (2002) to compute the worst-case performance for the flow shop case.

References

Aloulou, M. A. (2002) Structure flexible d'ordonnancements à performances contrôlées pour le pilotage d'atelier en présence de perturbations. *Ph.D. Thesis,* Institut National Polytechnique de Lorraine.

Aloulou, M. A., Kovalyov, M. Y. and Portmann, M. C. (2004) Maximization in single machine scheduling. *Annals of Operations Research*, **129**:21–32,

Aloulou, M. A. and Portmann, M. C. (2001) Incorporating flexibility in job sequencing for the single machine total weighted tardiness problem with release dates. In *Proceedings of 10th Annual Industrial Engineering Research Conference*, on CD-ROM.

Artigues, C., Roubellat, F. and Billaut, J.-C. (1999) Characterization of a set of schedules in a resource constrained multi-project scheduling problem with multiple modes. *International Journal of Industrial Engineering, Applications and Practice*, **6**:112–122.

Baker, K. R. (1974) *Introduction to Sequencing and Scheduling*. Wiley, New York.

Billaut, J. C. and Roubellat, F. (1996) A new method for workshop real time scheduling. *International Journal of Production Research*, **34**:1555–1579.

Brightwell, G. and Winkler, P. (1991) Counting linear extensions. *Order*, **8**:225–242.

Church, L. K. and Uzsoy, R. (1992) Analysis of periodic and event-drive rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, **5**:153–163.

Davenport, A. J. and Beck, J. C. (2000) A survey of techniques for scheduling with uncertainty. *http://www.eil.utoronto.ca/profiles/chris/chris.papers.html*.

Davis, L. (1991) *Handbook of Genetic Algorithms*. Van Nostrand-Reinhold, New York.

Djerid, L. and Portmann, M. C. (1996) Genetic algorithm operators restricted to precedent constraint sets: genetic algorithm designs with or without branch and bound approach for solving scheduling problems with disjunctive constraints. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics (Oct. 14–17)*, Vol. 4, pp. 2922–2927.

Herroelen, W. and Leus, R. (2003) Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, in press, available online 1 June 2004.

Jouglet, A. (2002) Ordonnancer sur une machine pour minimiser la somme des coût. *Ph.D. Thesis*, Université de Technologie de Compiègne.

Lawler, E. L. (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, **19**:544–546.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B. (1993) Sequencing and scheduling: algorithms and complexity. In *Logistics of Production and Inventory, Handbook in Operations Research and Management Science, Vol. 4*, Elsevier, Amsterdam, pp. 445–452.

Leon, V. J., Wu, S. D. and Storer, R. H. (1994) Robustness measures and robust scheduling for job shops. *IIE Transactions*, **26**:32–43.

Mehta, S. V. and Uzsoy, R. (1999) Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, **12**:15–38.

Morton, T. E. and Pentico, D. W. (1993) *Heuristic Scheduling with Apllications to Production Systems and Project Management*. Wiley, New York.

Portmann, M. C., Vignier, A., Dardilhac, C. D. and Dezalay, D. (1998) Branch and bound crossed with ga to solve hybrid flowshops. *European Journal of Operational Research*, **107**:389–400.

T'kindt, V. and Billaut, J. C. (2002) *Multicriteria Scheduling*. Springer, Berlin.

Vieira, G. E., Herrmann, J. W. and Lin, E. (2003) Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, **6**:35–58.

Wu, S. D. Byeon, E. S. and Storer, R. H. (1999) A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, **47**:113–124.

A DYNAMIC MODEL OF TABU SEARCH FOR THE JOB-SHOP SCHEDULING PROBLEM

Jean-Paul Watson
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110, USA
jwatson@sandia.gov

L. Darrell Whitley and Adele E. Howe
Computer Science Department
Colorado State University
Fort Collins, CO 80523, USA
{whitley,howe}@cs.colostate.edu

Abstract Although tabu search is one of the most effective meta-heuristics for solving the job-shop scheduling problem (JSP), very little is known about why this approach works so well and under what conditions it excels. Our goal is to develop models of tabu search algorithms for the JSP that answer these and other related research questions. We have previously demonstrated that the mean distance between random local optima and the nearest optimal solution, denoted $\overline{d}_{lopt\text{-}opt}$, is highly correlated with problem difficulty for a well-known tabu search algorithm for the JSP introduced by Taillard. In this paper, we discuss various shortcomings of the $\overline{d}_{lopt\text{-}opt}$ model and develop new models of problem difficulty that correct these deficiencies. We show that Taillard's algorithm can be modelled with exceptionally high fidelity using a surprisingly simple Markov chain. The Markov model also enables us to characterise the exact conditions under which different initialisation methods can be expected to improve performance. Finally, we analyse the relationship between the Markov and $\overline{d}_{lopt\text{-}opt}$ models.

Keywords: tabu search, job-shop, scheduling.

INTRODUCTION

Taillard (1989) first demonstrated the effectiveness of tabu search algorithms for the job-shop scheduling problem (JSP). Since then, researchers have introduced numerous improvements to Taillard's original algorithm (Blażewicz

et al., 1996). This effort has been rewarded: tabu search algorithms are widely regarded as among the most effective approaches for generating high-quality solutions to the JSP (Jain and Meeran, 1999). Yet, over the same time period, researchers have made little progress in developing a theoretical understanding of these algorithms. Specifically, little is known about why tabu search is so effective for the JSP and under what conditions we can expect such strong performance.

To gain insight into the behaviour of tabu search algorithms for the JSP, we recently performed an extensive analysis of the relationship between various features of the fitness landscape (i.e. the underlying search space) and search cost for Taillard's algorithm (Watson *et al.*, 2003; Watson, 2003). Our initial findings were largely negative: many features that are widely believed to influence problem difficulty, such as the number of optimal solutions, the backbone size (i.e. the number of solution attributes with identical values in all optimal solutions), fitness–distance correlation, and the mean distance between random local optima are in fact only weakly correlated with search cost. Drawing from research on problem difficulty for local search in MAX-SAT (the optimisation formulation of the Boolean Satisfiability decision problem), we showed that the mean distance between random local optima and the nearest optimal solution, which we denote $\overline{d}_{lopt\text{-}opt}$, is highly correlated with the cost required by Taillard's algorithm to locate optimal solutions to the JSP. We also demonstrated that $\overline{d}_{lopt\text{-}opt}$ accounts for much of both the variability in the difficulty of locating sub-optimal solutions and differences in the relative difficulty of square ($n/m = 1$) versus rectangular ($n/m > 1$) JSPs.

The $\overline{d}_{lopt\text{-}opt}$ model also has several shortcomings (Watson *et al.*, 2003). First, the model was developed and validated using small problem instances, leaving open the question of scalability. Second, despite good overall accuracy, model errors frequently exceed 1/2 an order of magnitude in search cost, and the model is least accurate for the most difficult problem instances. Third, the model provides no direct insight into the dynamic behaviour of the underlying search process: it is unclear *why* $\overline{d}_{lopt\text{-}opt}$ should be so highly correlated with search cost.

In this paper, we correct these deficiencies by introducing a dynamic model of problem difficulty that is based on an analysis of the run-time behaviour of Taillard's algorithm. In doing so, we make the following contributions toward a theoretical understanding of tabu search algorithms for the JSP:

1. The accuracy of the $\overline{d}_{lopt\text{-}opt}$ model can be significantly improved by considering the set of solutions visited by Taillard's algorithm *during* search.

2. Taillard's algorithm can be modelled with exceptionally high fidelity using a surprisingly simple Markov chain whose states represent both the current distance from the nearest optimal solution and the current

search gradient, i.e. whether search is progressing toward or away from the nearest optimal solution. The Markov model accounts for nearly all of the variability in the cost required to locate optimal solutions to both small (6×4, 6×6) and large (10×10) random JSPs. The model also provides insight into the exact conditions under which different initialisation methods can be expected to improve performance.

3. We identify a relationship between the Markov and $\overline{d}_{lopt\text{-}opt}$ models, which enables us to account for why $\overline{d}_{lopt\text{-}opt}$ is so highly correlated with search cost.

The remainder of this paper is organised as follows. First, in Section 1, we summarise the key features of Taillard's algorithm and discuss our experimental methodology. We summarise the key features of the $\overline{d}_{lopt\text{-}opt}$ model in Section 2 and discuss several deficiencies of the model. In Section 3, we show that the mean distance between solutions visited during search and the nearest optimal solution is more highly correlated with search cost than $\overline{d}_{lopt\text{-}opt}$. We develop a dynamic Markov model of the behaviour of Taillard's algorithm in Section 4 and use the resulting model to explore the conditions under which heuristic initialisation can be expected to improve performance in Section 5. We conclude by discussing the implications of our results in Section 6.

1. PROBLEM, ALGORITHM, AND METHODOLOGY

We consider the well-known $n \times m$ static JSP (Blażewicz *et al.*, 1996) in which n jobs must be processed exactly once on each of m machines. Each job i ($1 \leq i \leq n$) is routed through each of the m machines in a pre-defined order π_i, where $\pi_i(j)$ denotes the jth machine ($1 \leq j \leq m$) in the routing order. The processing of job i on machine $\pi_i(j)$ is denoted o_{ij} and is called an operation. An operation o_{ij} must be processed on machine $\pi_i(j)$ for an integral duration $\tau_{ij} \geq 0$. Once processing is initiated, an operation cannot be pre-empted, and concurrency is not allowed. For $2 \leq j \leq m$, o_{ij} cannot initiate processing until o_{ij-1} has completed. The scheduling objective is makespan minimisation, i.e. to minimise the maximum completion time of the last operation of any job.

An instance of the $n \times m$ JSP is defined by the set of nm operation durations τ_{ij} and n job routing orders π_i. We define a *random* JSP as an instance generated by (1) sampling the τ_{ij} independently and uniformly from a fixed-width interval and (2) constructing the π_i from random permutations of the integers $[1..m]$. Most well-known JSP benchmark instances (available from the OR Library: www.ms.ic.ac.uk/info.html) are random JSPs; exceptions include instances such as `swv11-15` in which the π_i are more constrained (e.g., they possess workflow or flowshop partitions).

Our analysis is based on a tabu search algorithm for the JSP introduced by Taillard (1989, 1994), which we denote $TS_{Taillard}$. We observe that $TS_{Taillard}$ is

not the best available tabu search algorithm for the JSP: the algorithms of Nowicki and Smutnicki (1996) and Chambers and Barnes (1996) provide stronger overall performance. We chose $TS_{Taillard}$ because it is particularly amenable to analysis (for reasons discussed below) and serves as a basis for more advanced algorithms. Relative to $TS_{Taillard}$, state-of-the-art tabu search algorithms for the JSP employ more effective move operators and frequently re-intensify search around high-quality solutions. Instead of tackling state-of-the-art algorithms, our (pragmatic) approach is to first develop models of a basic algorithm ($TS_{Taillard}$) and to subsequently extend this model to account for algorithmic features found in state-of-the-art algorithms.

$TS_{Taillard}$ is a relatively straightforward implementation of tabu search and is based on the well-known *N1* move operator introduced by van Laarhoven *et al.* (1992). The neighbourhood of a solution s under *N1* consists of the set of solutions obtained by inverting the order of a pair of adjacent critical operations on the same machine. Taillard's original papers (Taillard, 1989; Taillard, 1994) provide a detailed description of $TS_{Taillard}$. A key feature of $TS_{Taillard}$ is the dynamic tabu tenure, which is periodically and uniformly re-sampled from a fixed-width interval $[L_{min}, L_{max}]$. The combination of a dynamic tabu tenure (which prevents cycling) and the *N1* move operator (which guarantees that a path exists from an arbitrary solution to some optimal solution, see van Laarhoven *et al.*, 1992) endows $TS_{Taillard}$ with a key property: for reasonable values of L_{min} and L_{max}, the algorithm is (at least empirically) guaranteed to eventually locate an optimal solution (see our discussion in Watson *et al.*, 2003).

Our results are based on our own implementation of $TS_{Taillard}$, which deviates from Taillard's original algorithm in three respects. First, instead of initiating search from a lexicographic solution (constructed by scheduling the jobs in index order), we use a local optimum generated by applying steepest-descent (under the *N1* operator) to a random semi-active solution (Mattfeld *et al.*, 1999); we investigate alternate initialisation methods in Section 5. Second, we compute the makespan of neighbouring solutions exactly; Taillard's original algorithm employed an estimation scheme. Third, we do not use the long-term frequency-based memory mechanism introduced by Taillard. Our modifications enable us to control for any possible impact of these mechanisms on the development of accurate behavioural models of $TS_{Taillard}$.

The key behaviour of interest when analysing $TS_{Taillard}$ is the cost required to locate *optimal* solutions to problem instances. For individual trials, this cost is naturally defined as the number of iterations c required to locate an optimal solution. Because $TS_{Taillard}$ is stochastic (due to the choice of initial solution, random tie-breaking when multiple "best" moves are available, and the tabu tenure), search cost is in fact a random variable with an approximately exponential distribution (Taillard, 1994). Consequently, we run 1,000 independent

trials of $TS_{Taillard}$ for a given instance, and define search cost as either the mean ($\bar{c}$) or median (c_{Q2}) number of iterations required to locate an optimal solution, depending on the context (estimates of c_{Q2} are less sensitive to extreme values and are used when possible).

As in our previous research (Watson *et al.*, 2003), we develop and validate our models of $TS_{Taillard}$ using sets of 6×4 and 6×6 random JSPs containing 1,000 instances each, with the τ_{ij} uniformly sampled from the interval $[1, 99]$. Additionally, due to recent advances in computing power, we are now able to assess model scalability using a set of one hundred 10×10 random JSPs (also generated by sampling the τ_{ij} uniformly from the interval $[1, 99]$). We also consider the following well-known 10×10 random JSP benchmark instances: `la16-la20` and `abz5-abz6`. We ignore several other well-known instances (e.g., Fisher and Thompson's infamous 10×10 instance and `orb01-orb10`) because the job routing orders are far more structured than that of a typical random JSP. We are unable to consider larger rectangular problem instances (i.e. instances with $n \gg m$) due to the large number of optimal solutions. For experiments involving our 6×4 and 6×6 problem sets, we set the L_{min} and L_{max} parameters of $TS_{Taillard}$ to 6 and 14, respectively; for our 10×10 problem set, we let $L_{min} = 8$ and $L_{min} = 14$. The models we develop are functions of the set of *all* optimal solutions to a given problem instance. We use Beck and Fox (2000)'s constraint programming algorithm to both compute the optimal makespan and to enumerate the set of optimal solutions for our test instances. Our models are also based on the notion of distance between pairs of solutions, which we take as the well-known *disjunctive graph* distance: see Mattfeld *et al.* (1999). Informally, the disjunctive graph distance between two solutions is the number of differences in the relative order of distinct pairs of operations on the same machine.

2. A SUMMARY AND CRITIQUE OF PRIOR RESULTS

Previously, we analysed the relationship between various fitness landscape features and problem difficulty for $TS_{Taillard}$ (Watson *et al.*, 2003; Watson, 2003). We used regression methods to develop statistical models relating one or more of these features to the cost $\log_{10}(c_{Q2})$ required to locate optimal solutions to 6×4 and 6×6 random JSPs. Because they are based on static (i.e. independent of meta-heuristic) features of the fitness landscape, we refer to these models as *static cost models*. The accuracy of a static cost model can be quantified as the model r^2, i.e. the proportion of the total variability in search cost accounted for by the model. We found that the accuracy of static cost models based on well-known features such as the number of optimal solutions, the

backbone size, and the average distance between random local optima is only weak-to-moderate, with r^2 ranging from 0.22 to 0.53.

Drawing from research on problem difficulty for local search in MAX-SAT (Singer *et al.*, 2000), we demonstrated that a static cost model based on the mean distance between random local optima and the nearest optimal solution, which we denote $\overline{d}_{lopt\text{-}opt}$, is significantly more accurate, yielding r^2 values of 0.8260 and 0.6541 for 6×4 and 6×6 random JSPs, respectively. The $\overline{d}_{lopt\text{-}opt}$ model is a descriptive model indicating that search cost is largely an exponential function of the total distance that must be traversed, i.e. between the initial (random) solution and the nearest optimal solution. For illustrative purposes, we provide a scatter-plot of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} in the left side of Figure 1 for 6×6 random JSPs. The actual c_{Q2} are typically within an order of magnitude of the predicted c_{Q2}; in a few exceptional cases, the error can reach two orders of magnitude. Additionally, we showed that $\overline{d}_{lopt\text{-}opt}$ accounts for most of the variability in the cost required to locate *sub-optimal* solutions, i.e. with makespans larger than the global minimum, to these same problem instances (providing an explanation for the existence of cliffs in the search cost at particular offsets from the optimal makespan) and differences in the relative difficulty of square ($n/m = 1$) versus rectangular ($n/m > 1$) problem instances.

We also identified several deficiencies of the $\overline{d}_{lopt\text{-}opt}$ model. First, as shown in the left side of Figure 1, there is evidence that model accuracy is inversely proportional to $\overline{d}_{lopt\text{-}opt}$, i.e. the magnitude of the average residual at a particular value of $\overline{d}_{lopt\text{-}opt}$ is proportional to $\overline{d}_{lopt\text{-}opt}$. Of particular concern is the fact that the model is least accurate for the most difficult problem instances. Second, the model fails to account for a non-trivial proportion ($\approx 1/3$) of the variability in problem difficulty for the 6×6 instances. Third, the differences in accuracy observed for the 6×4 and 6×6 instances raises concerns regarding scalability of the model to larger, more realistically sized problem instances. Fourth, and perhaps most importantly, the model provides little direct insight as to *why* the mean distance between random local optima and the nearest optimal solution should be so highly correlated with search cost.

To assess the scalability of the $\overline{d}_{lopt\text{-}opt}$ model, we computed $\overline{d}_{lopt\text{-}opt}$ for the 92 instances of our 10×10 problem set with $\leq$50 million optimal solutions; the computational costs are currently prohibitive for the remaining eight instances. For each instance, we use 5,000 random local optima (generated by applying steepest-descent to random semi-active solutions) to estimate $\overline{d}_{lopt\text{-}opt}$. We show a scatter-plot of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} for these instances in the right-hand side of Figure 1. The r^2 value for the corresponding regression model is 0.4598, a 33% decrease in model accuracy relative to the 6×6 problem set. This result clearly demonstrates the failure of the $\overline{d}_{lopt\text{-}opt}$ to scale to larger problem instances.

One obvious extension to our research would involve the development and analysis of more complex static cost models: e.g., those based on multiple

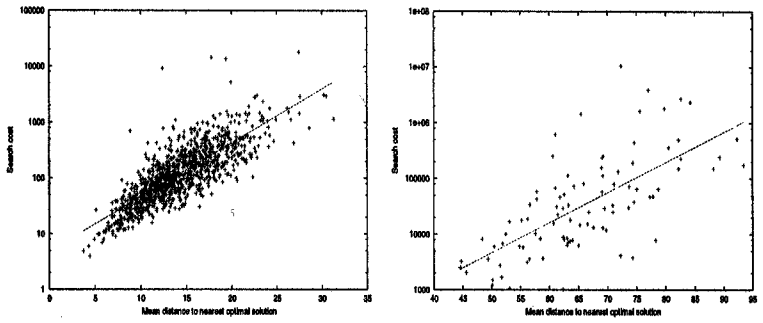


Figure 1. Scatter-plots of $\overline{d}_{lopt\text{-}opt}$ versus c_{Q2} for 6×6 (left figure) and 10×10 (right figure) random JSPs; the least-squares fit lines are superimposed.

features or features that capture more detail of the fitness landscape. However, we found in pilot experiments that more complex models appear to yield at best only marginal (less than 5%) improvements in model accuracy. This result is consistent with our intuition: static cost models explicitly *ignore* the dynamic behaviour of $TS_{Taillard}$. To develop truly accurate (i.e. $r^2 \geq 0.9$) cost models, we thought it necessary to account for the dynamic behaviour of the algorithm under consideration. We now explicitly test this hypothesis, by developing both simple (Section 3) and complex (Section 4) cost models that are functions of the set of solutions visited by $TS_{Taillard}$ *during* search.

3. THE IMPACT OF SEARCH BIAS

Attractor basins of local optima in the JSP are surprisingly weak (i.e. narrow and shallow): they can be escaped with high probability by accepting one or two dis-improving moves and re-initiating greedy descent (Watson, 2003). Thus, search under $TS_{Taillard}$ is generally restricted to the sub-space of solutions containing both local optima and solutions very close (in terms of distance) to local optima. The intuition behind $\overline{d}_{lopt\text{-}opt}$ is that it represents the *effective* size of this sub-space (by taking into account the number of optimal solutions) and as a consequence is highly correlated with search cost. However, the deficiencies of the $\overline{d}_{lopt\text{-}opt}$ static cost model (specifically, the lack of scalability) suggest that either (1) $\overline{d}_{lopt\text{-}opt}$ is not an entirely accurate indicator of the size of the local optima sub-space or (2) the size of the local optima sub-space is not completely indicative of search cost. We now focus on the first alternative, by developing a more accurate measure of the size of the local optima sub-space.

The $\overline{d}_{lopt\text{-}opt}$ measure is a function of the distribution of the distance between *random* local optima and the nearest optimal solution (d_{opt}). Consider instead the distribution of d_{opt} for solutions visited by $TS_{Taillard}$ *during* search. Both

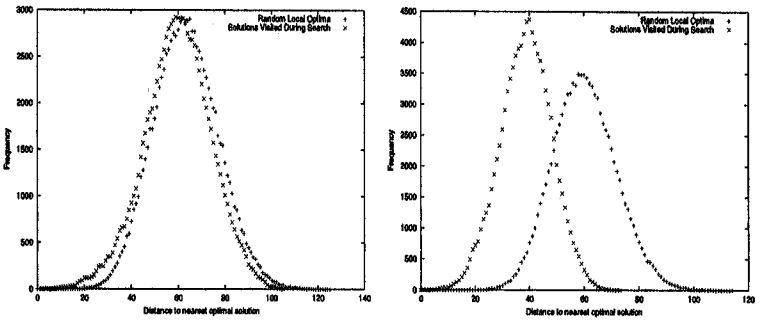


Figure 2. Histograms of the distance to the nearest optimal solution (d_{opt}) for (a) 100,000 random local optima and (b) 100,000 solutions generated by $TS_{Taillard}$ for two 10×10 random JSPs.

distributions are typically Gaussian-like, although we have observed skewed distributions both with and without heavy tails. We provide examples of these distributions for two 10×10 instances in Figure 2. Although the two distributions are often identical (especially in 6×4 instances and to a lesser extent in 6×6 instances), they can also possess very different means and variances, as shown in the right side of Figure 2. Further, we commonly observe such discrepancies in our 10×10 instances—the same instances for which the $\overline{d}_{lopt\text{-}opt}$ model is least accurate.

These observations led us to conjecture that the mean d_{opt} for solutions visited during search, which we denote $\overline{d}_{tabu\text{-}opt}$, may be a more accurate indicator of the size of the local optima sub-space than $\overline{d}_{lopt\text{-}opt}$. For a given instance, we compute $\overline{d}_{tabu\text{-}opt}$ using a set of 100,000 solutions visited by $TS_{Taillard}$ over a variable number of independent trials. Each trial is initiated from a random local optimum and terminated once an optimal solution is located; we impose the termination criterion because there exist optimal solutions from which no moves are possible under the *N1* operator (Nowicki and Smutnicki, 1996). We terminate the entire process, including the current trial, once we obtain 100,000 samples.

Regression models of $\overline{d}_{tabu\text{-}opt}$ versus $\log_{10}(c_{Q2})$ yielded r^2 values of 0.8441 for our 6×4 instances and 0.7808 for our 6×6 instances; this corresponds to roughly 4% and 20% increases in accuracy over that of the $\overline{d}_{lopt\text{-}opt}$ model, respectively. The scatter-plot for the 6×6 instances is shown in the left-hand side of Figure 3. In either case, the absolute accuracy is remarkably high. The variable impact on accuracy is due to frequency of instances in which the distributions of d_{opt} for random local optima and solutions visited during search are dissimilar. The actual c_{Q2} typically deviate from the predicted c_{Q2} by no more than $1/2$ an order of magnitude, and we observe fewer and less extreme

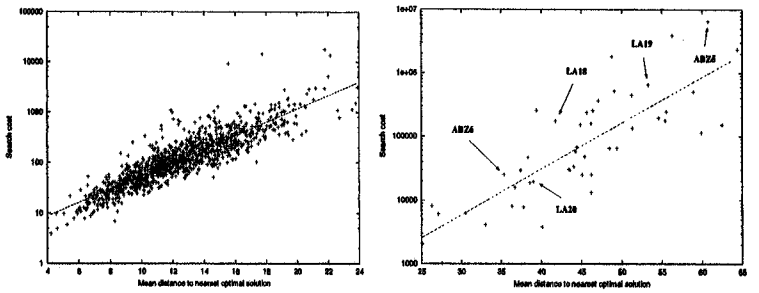


Figure 3. Scatter-plots of $\overline{d}_{tabu\text{-}opt}$ versus search cost (c_{Q2}) for 6×6 (left figure) and 10×10 (right figure) random JSPs; the least-squares fit lines are superimposed.

high-residual instances than under the $\overline{d}_{lopt\text{-}opt}$ model. Finally, although beyond the scope of this paper, the $\overline{d}_{tabu\text{-}opt}$ model provides similar improvements in the ability to predict the cost of locating sub-optimal solutions to these same problem instances.

We next assess the accuracy of the $\overline{d}_{tabu\text{-}opt}$ model on the set of forty-two 10×10 instances with $\leq$100,000 optimal solutions; the computation of $\overline{d}_{tabu\text{-}opt}$ is prohibitively expensive for the remaining instances. Given the relatively poor correlation between the number of optimal solutions and search cost (Watson *et al.*, 2003), our selection criterion does *not* lead to a clean distinction between easy and hard problem instances; the hardest instance in our 10×10 problem set has approximately 1.5 million optimal solutions. However, on average, instances with $\leq$100,000 optimal solutions are generally more difficult, with a median c_{Q2} of 65,710, versus 13,291 for instances with more than 100,000 optimal solutions.

A regression model of $\overline{d}_{tabu\text{-}opt}$ versus $\log_{10}(c_{Q2})$ for these 10×10 instances yielded an r^2 value of 0.6641; we show the corresponding scatter-plot in the right-hand side of Figure 3. The resulting r^2 represents an approximately 41% increase in accuracy over the $\overline{d}_{lopt\text{-}opt}$ model. The model residuals typically vary from between $1/2$ and 1 order of magnitude, leaving a moderate proportion of the variability in search cost unexplained. The difference in model r^2 between the 6×6 and 10×10 problem sets is only $\approx$0.14. We have also annotated the scatter-plot with data for five of the seven 10×10 random JSPs found in the OR Library; both `la16` and `la17` have approximately 6.8 and 11.8 million optimal solutions, respectively, making computation of $\overline{d}_{tabu\text{-}opt}$ prohibitive. The `abz5` and `la19` instances are known to be significantly more difficult than their respective counterparts (i.e. `abz6`, `la18`, and `la20`) for numerous local search algorithms (Jain and Meeran, 1999), which is consistent given the observed differences in $\overline{d}_{tabu\text{-}opt}$.

Our results clearly demonstrate that the mean distance between solutions visited *during* search and the nearest optimal solution ($\overline{d}_{tabu\text{-}opt}$) is highly correlated with the cost required by Taillard's algorithm to locate optimal solutions to random JSPs. However, two key issues remain. First, as shown by the difference in r^2 obtained for the 6×6 and 10×10 instances, there is still some evidence that the $\overline{d}_{tabu\text{-}opt}$ model may fail to scale to even larger problem instances. Second, as was the case with $\overline{d}_{lopt\text{-}opt}$, it is unclear *why* $\overline{d}_{tabu\text{-}opt}$ is so highly correlated with search cost. To address these issues, we now examine the dynamic behaviour of $TS_{Taillard}$ in more detail.

4. A DYNAMIC COST MODEL OF TABU SEARCH

The dynamic behaviour of memoryless local search algorithms (e.g., simulated annealing or iterated local search) can, at least in principle, be modelled using Markov chains: the set of feasible solutions is known and the transition probabilities between neighbouring solutions can be computed. Although exact, such models require an exponential number of states for most interesting (i.e. NP-hard) problems and therefore provide little insight into the qualitative nature of the search process. The challenge is to develop lumped models, in which large numbers of solutions are grouped into individual states, yielding more tractable and consequently understandable Markov chains. A similar approach is possible when modelling tabu search, although we must additionally embed the contents of short-term memory into the state definition. We then face two questions: "What criterion do we use to aggregate solutions?" and "How do we model short-term memory?"

Given the objective of makespan minimisation, we aggregate solutions based on their distance to the nearest optimal solution. To model the impact of short-term memory, we analyse how search progresses in terms of consistent trends either toward or away from the nearest optimal solution. In Figure 4, we show a time-series of the distance to the nearest optimal solution for a random walk (left figure) and $TS_{Taillard}$ (right figure) for a 10×10 random JSP; these particular time-series were selected to yield figures with identical ranges in the distance to the nearest optimal solution. As expected, the random walk exhibits minimal trending behaviour. In contrast, $TS_{Taillard}$ exhibits distinct trending behaviour, often maintaining the current search gradient for extended periods of time. Similar results hold in a limited sampling of our 6×4, 6×6, and 10×10 instances. This suggests that $TS_{Taillard}$'s short-term memory mechanism influences the search process simply by consistently biasing search either toward or away from the nearest optimal solution.

Given strong trending behaviour, we define a state $S_{i,grad}$ in our Markov model as a pair representing (1) the set of solutions distance i from the nearest optimal solution and (2) the current search gradient *grad*. We define *grad* as

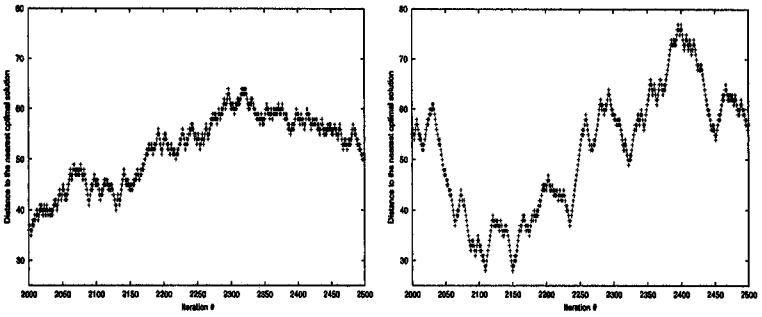


Figure 4. Time-series of the distance to the nearest optimal solution for the solutions visited by a random walk (left figure) and $TS_{Taillard}$ (right figure) for a typical 10×10 random JSP.

the difference in d_{opt} for solutions from the current and immediately preceding iteration of $TS_{Taillard}$. Although $grad \in \{-1, 0, 1\}$, for clarity we denote these numeric values symbolically by *closer*, *equal*, and *farther*, respectively. In effect, we are modelling the impact of short-term memory as a *single* scalar (*grad*) and embedding this scalar into the state definition. Given a maximum possible distance of D from a solution to the nearest optimal solution, our Markov model consists of exactly $3 \cdot (D + 1)$ states (the "+1" state represents the set of optimal solutions).

Next, we specify the transition probabilities between pairs of states $S_{i,grad'}$ and $S_{j,grad}$ in our model. We let the conditional probability $P(S_{i,grad'}|S_{j,grad})$ denote the probability of *simultaneously* altering the search gradient from *grad* to *grad'* and moving from a solution at distance j from the nearest optimal solution to a solution at distance i away from the nearest optimal solution. The majority of these probabilities obviously equal 0: specifically, for any pair of states $S_{i,grad'}$ and $S_{j,grad}$ with $|i - j| > 1$, or when simultaneous changes in both gradient and distance to the nearest optimal solution are logically impossible, such as from state $S_{i,closer}$ to state $S_{i+1,closer}$. For each i such that $1 \leq i \leq D$, exactly nine non-zero transition probabilities are possible:

$$P(S_{i-1,closer}|S_{i,closer}), P(S_{i,equal}|S_{i,closer}), P(S_{i+1,farther}|S_{i,closer})$$
$$P(S_{i-1,closer}|S_{i,equal}), P(S_{i,equal}|S_{i,equal}), P(S_{i+1,farther}|S_{i,equal})$$
$$P(S_{i-1,closer}|S_{i,farther}), P(S_{i,equal}|S_{i,farther}) \text{ and } P(S_{i+1,farther}|S_{i,farther})$$

The set of transition probabilities is also subject to the total-probability constraints:

$$P(S_{i-1,closer}|S_{i,closer}) + P(S_{i,equal}|S_{i,closer}) + P(S_{i+1,farther}|S_{i,closer}) = 1$$
$$P(S_{i-1,closer}|S_{i,equal}) + P(S_{i,equal}|S_{i,equal}) + P(S_{i+1,farther}|S_{i,equal}) = 1$$

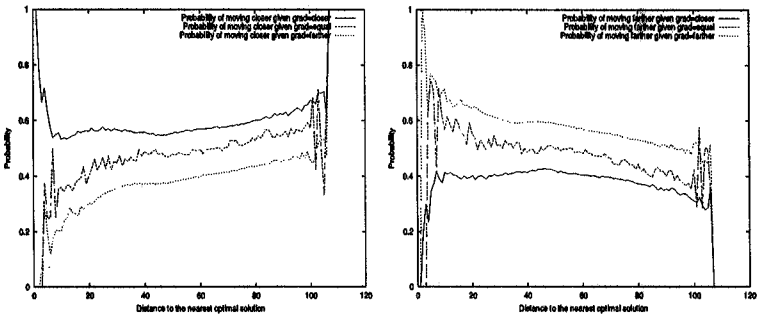


Figure 5. The transition probabilities for moving closer to (left figure) or farther from (right figure) the nearest optimal solution under $TS_{Taillard}$ for a typical 10×10 random JSP.

$$P(S_{i-1,closer}|S_{i,farther}) + P(S_{i,equal}|S_{i,farther}) + P(S_{i+1,farther}|S_{i,farther}) = 1$$

In order to complete our Markov model, we impose a reflective barrier at $i = D$ and an absorbing state at $i = 0$ by imposing the constraints $P(S_{D+1,farther}|S_{D,farther}) = 0$ and $P(S_{0,equal}|S_{0,equal}) = 1$, respectively.

We estimate the set of transition probabilities for a given problem instance by analysing the set of solutions visited by $TS_{Taillard}$ over a large number of independent trials. At each iteration $k \geq 1$ of each trial, we compute (1) the distance j from the current solution s to the nearest optimal solution and (2) the current search gradient *grad*, which is a function of solutions encountered in both the current (kth) and previous, ($k-1$)th, iterations. Given the neighbour s' of s selected for the next, ($k+1$)th, iteration, we then compute (3) the distance i from s' to the nearest optimal solution and (4) the search gradient *grad'* given the solutions s' and s. We denote the total number of occurrences of state $S_{j,grad}$ by $\#(S_{j,grad})$ and the total number of occurrences of a successor state $S_{i,grad'}$ given a current state $S_{j,grad}$ by $\#(S_{i,grad'}|S_{j,grad})$; both quantities are tracked over all iterations of all trials. We execute $TS_{Taillard}$ until $\#(S_{i,closer}) \geq 50$ for $1 \leq i \leq rint(\bar{d}_{lopt\text{-}opt})$. Individual trials are initiated from random local optima and terminated once an optimal solution is located; the *rint* function returns the integer nearest the input, rounding up when the fractional component equals 0.5. Because $TS_{Taillard}$ is empirically guaranteed to eventually locate an optimal solution and there is a non-zero probability of initiating a trial from a random local optimum that is distance $i \geq rint(\bar{d}_{lopt\text{-}opt})$ from the nearest optimal solution, the termination criterion will eventually be satisfied as the number of trials approaches ∞.

We compute estimates of the transition probabilities using the obvious formulae, e.g. $P(S_{i-1,closer}|S_{i,closer}) = \#(S_{i-1,closer}|S_{i,closer})/\#(S_{i,closer})$. For $i > rint(\bar{d}_{lopt\text{-}opt})$, $\#(S_{i,closer}) \geq 50$ is relatively common. Consequently, we take $D = X - 1$ where X is the minimal value satisfying $\#(S_{X,closer}) < 50$.

Empirically, omitting states $S_{i,grad}$ with $i \gg \overline{d}_{lopt\text{-}opt}$ has negligible impact on model accuracy. Estimates of the transition probabilities are largely insensitive to both the random initial local optima and the sequence of solutions visited during individual trials, i.e. the statistics appear to be isotropic.

In Figure 5, we show the estimated probabilities of moving closer to (left figure) or farther from (right figure) the nearest optimal solution for a typical 10×10 random JSP; the probability of maintaining an *equal* search gradient is negligible ($p < 0.1$) for all i. We observe qualitatively similar results for all of our test instances. These results indicate that tabu search in the JSP can be viewed as a diffusion process with a central restoring force—the probability of moving closer to (farther from) the nearest optimal solution is proportional (inversely proportional) to the current distance from the nearest optimal solution. In other words, there is pressure toward solutions that are equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution, which is consistent with the Gaussian-like histograms of d_{opt} observed for $TS_{Taillard}$ (as shown in Figure 2). The impact of short-term memory is also evident, in that the probability of continuing to move along the current gradient is very strong and exceeds 0.5 independently of i for nearly all problem instances. This bias accounts for the strong trending behaviour observed in the time-series of d_{opt} for $TS_{Taillard}$, as shown in the right-hand side of Figure 4. Finally, in contrast to the transition probabilities under a random walk, as $i \rightarrow 0$ the probability of continuing to move closer to the optimal solution actually *rises*, typically approaching 1 at some $i \geq 5$.

To validate our Markov model, we compare the predicted versus actual mean search cost $\overline{c}$ for our 6×4, 6×6, and 10×10 problem sets. For a given instance, we estimate the predicted $\overline{c}$ by repeatedly simulating the Markov chain defined by D, the set of states $S_{i,grad}$, and the estimated transition probabilities $P(S_{i,grad'}|S_{j,grad})$; analytic results for mean time-to-absorption are, to the best of our knowledge, unavailable for this form of Markov chain in the general case. Let v_i denote the mean number of iterations (with statistics taken over 10,000 samples) required to achieve a state $S_{0,closer}$ given an initial state $S_{i,X}$. We set the initial value of X to equal either *closer* or *farther* with equal probability, given that the probability of maintaining an *equal* search gradient is negligible. Let $\delta = rint(\overline{d}_{lopt\text{-}opt})$. We define the predicted mean search cost ($\overline{c}$) as v_δ: i.e. search is initiated from solutions that are, on average, roughly distance $\overline{d}_{lopt\text{-}opt}$ from the nearest optimal solution.

For our 6×4 and 6×6 instances, $\log_{10}$–$\log_{10}$ regression models (i.e. in which the $\log_{10}$ transformation is applied to both the independent and dependent variables) of the predicted versus actual $\overline{c}$ yielded r^2 values of 0.9941 and 0.9939, respectively; we show the corresponding scatter-plot for the 6×6 instances in the left-hand side of Figure 6. Model accuracy is exceptionally high in both problem sets, and for all instances the predicted $\overline{c}$ is within a factor of 2

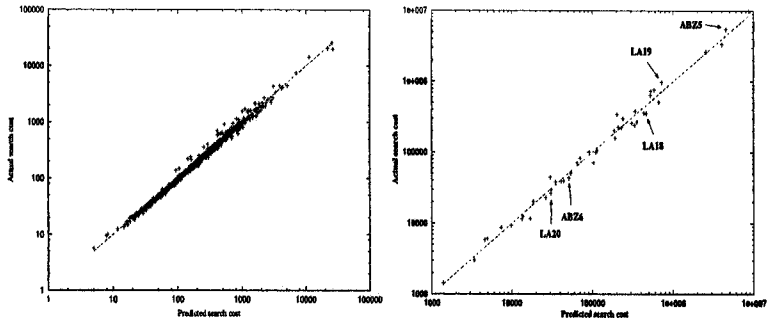


Figure 6. Scatter-plots of the observed versus predicted mean cost ($\overline{c}$) to locate an optimal solution under $TS_{Taillard}$ for 6×6 (left figure) and 10×10 (right figure) random JSPs; the least-squares fit lines are superimposed.

of the actual $\overline{c}$. To assess the scalability of our Markov model, we consider the subset of forty-two 10×10 instances with $\leq 100,000$ optimal solutions; the estimation of the transition probabilities is computationally prohibitive for the remaining instances. For these instances, a $\log_{10}$–$\log_{10}$ regression model of the predicted versus actual $\overline{c}$ yielded an r^2 value of 0.9877. We show the corresponding scatter-plot in the right-hand side of Figure 6, which additionally includes the results for the benchmark instances `la18-la20` and `abz5-abz6`. These results clearly demonstrate that the behaviour of Taillard's algorithm can be modelled with high fidelity as a simple one-dimensional random walk. In contrast to both the $\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{tabu\text{-}opt}$ models, the Markov model appears scalable. Finally, we note that our Markov model is equally successful in accounting for the variability in the cost of locating sub-optimal solutions to these same problem instances.

4.1 The Relationship Between the Models

In hindsight, the success of the $\overline{d}_{lopt\text{-}opt}$ model was due to the fact that $\overline{d}_{lopt\text{-}opt}$ and $\overline{d}_{tabu\text{-}opt}$ are highly correlated for small problem instances. What remains is to establish a link between the $\overline{d}_{tabu\text{-}opt}$ model and the Markov model. As previously noted, the *qualitative* forms of the estimated transition probabilities (e.g., see Figure 5) are identical for all of the problem instances we examined. Any major differences are due to variability in D, which (like $\overline{d}_{tabu\text{-}opt}$) can be viewed as a measure of the size of the local optima sub-space. We also observe that these transition probabilities are roughly symmetric around $D/2$ and that search in $TS_{Taillard}$ is necessarily biased toward solutions that are approximately distance $D/2$ from the nearest optimal solution. But the latter quantity is essentially equivalent to $\overline{d}_{tabu\text{-}opt}$, and consequently $\overline{d}_{tabu\text{-}opt} \approx D/2$. Thus, we

believe the success of the $\overline{d}_{tabu\text{-}opt}$ model is due to the fact that it estimates a key parameter (D) of the Markov model.

5. THE IMPACT OF INITIALISATION METHOD ON PERFORMANCE

Our models of $TS_{Taillard}$ are based on the assumption that search is initiated from a random local optimum. But can our models yield any insight into the impact of heuristic initialisation on algorithm performance? Although researchers generally agree that high-quality initial solutions *can* improve the performance of tabu search algorithms (e.g., see Jain *et al.*, 2000), the exact conditions under which improvements can be achieved, and the expected degree of improvement, are poorly understood. In this section, we explore a particular aspect of this broader issue: What impact do different initialisation methods, both heuristic and random, have on the cost required by $TS_{Taillard}$ to locate *optimal* solutions to problem instances?

To validate our Markov model, we only computed v_δ: the mean number of iterations required to locate an optimal solution, given a starting point that is (modulo rounding) distance $\overline{d}_{lopt\text{-}opt}$ from the nearest optimal solution. But what does our model predict if search is initiated from a solution that is either closer to or farther from the nearest optimal solution than δ? In Figure 7, we show plots of the predicted costs v_i over the full range of i for a 6×6 (left figure) and 10×10 (right figure) random JSP. For the 6×6 instance, search cost rises rapidly between $i = 3$ and $i = 10$, but only gradually increases once $i > 10$. In contrast, search cost for the 10×10 instance rises rapidly between $i = 2$ and $i \approx 15$, but is roughly *constant* (modulo the sampling noise) once $i > 15$. Even when $i = 2$, our model predicts that search cost is still significant: if the initial search gradient is not *closer*, search is driven toward solutions that are distant from the nearest optimal solution and any benefit of a favourable initial position is lost. We observed qualitatively identical behaviour in a large sampling of our problem instances: for easy (hard) instances, the approach toward an asymptotic value as $i \rightarrow D$ is gradual (rapid).

Our Markov model predicts that the distance to the nearest optimal solution, and not the fitness, dictates the benefit of a particular initialisation method. The distinction is especially key in the random JSP, where fitness–distance correlation is known to be comparatively weak (Mattfeld, 1996). In particular, our model predicts that an initialisation method will at best have a minimal impact on search cost *unless* the resulting solutions are very close to the nearest optimal solution. To test this hypothesis, we analysed the performance of $TS_{Taillard}$ using a variety of heuristic and random initialisation methods. Following Jain *et al.* (2000), we consider the following set of high-quality priority dispatch rules (PDRs), used in conjunction with Giffler and Thompson's procedure for

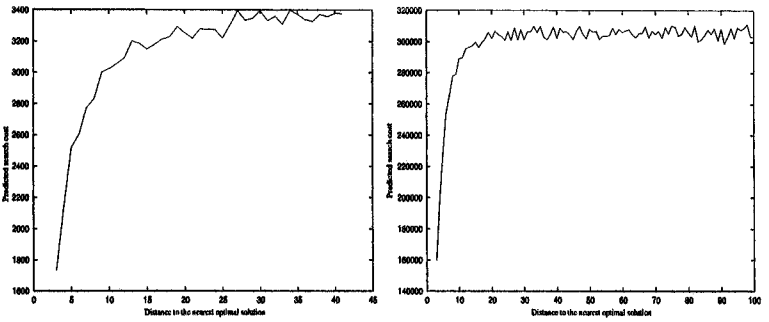


Figure 7. Predicted cost required by $TS_{Taillard}$ to locate an optimal solution, given an initial solution that is distance i from the nearest optimal solution, for a 6×6 (left figure) and a 10×10 (right figure) random JSP.

generating active schedules (Giffler and Thompson, 1960): *FCFS* (First-Come, First-Serve), *LRM* (Longest ReMaining work), *MWKR* (Most WorK Remaining), and *SPT* (Shortest Processing Time). We additionally considered both active and non-delay solutions (Giffler and Thompson, 1960) generated using random PDRs, which we respectively denote RND_{active} and $RND_{nondelay}$. We denote our baseline random semi-active solutions by $RND_{semiactive}$. Finally, we examined Taillard's original lexicographic solution method, denoted *LEXICO*, and the insertion procedure introduced by Werner and Winkler (1995), which we denote *WW*; the latter is one of the best constructive heuristics available for the random JSP (Jain *et al.*, 2000). As with $RND_{semiactive}$, we post-process the resulting solutions by applying steepest-descent under the *N1* operator to generate a local optimum.

For each initialisation method, we computed $\overline{d}_{lopt\text{-}opt}$ (by substituting the optima generated by a particular initialisation method for random local optima) for the forty-two 10×10 instances with $\leq$100,000 optimal solutions. With the exception of *LEXICO*, all of the methods we consider are stochastic. Consequently, we define $\overline{d}_{lopt\text{-}opt}$ as the mean distance between 5,000 random solutions and the nearest optimal solution. We show the resulting $\overline{d}_{lopt\text{-}opt}$ for each initialisation method in Table 1. We also provide the p-values for the statistical significance of the mean difference in $\overline{d}_{lopt\text{-}opt}$ between the various methods and $RND_{semiactive}$, which we obtained using a Wilcoxon signed rank test. With the exception of *SPT*, we observe significant differences in $\overline{d}_{lopt\text{-}opt}$ between our baseline method ($RND_{semiactive}$) and the other initialisation methods. Initially, these data appear to suggest that it may be possible to improve the performance of $TS_{Taillard}$ using initialisation methods with low $\overline{d}_{lopt\text{-}opt}$. However, the lowest absolute values of $\overline{d}_{lopt\text{-}opt}$ (obtained using the *LEXICO* and *WW* methods) are still large. For our 10×10 instances, our Markov model predicts that

Table 1. The differences in both the mean distance to the nearest optimal solution ($\bar{d}_{lopt\text{-}opt}$) and search cost ($c_{Q2}$) for various initialisation methods, measured relative to random semi-active solutions ($RND_{semiactive}$).

Initialisation method				
FCFS	*LRM*	*MWKR*	*SPT*	$RND_{semiactive}$
		$\bar{d}_{lopt\text{-}opt}$		
58.49	97.41	97.94	64.97	70.92
Significance of mean difference in $\bar{d}_{lopt\text{-}opt}$ relative to $RND_{semiactive}$				
$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p = 0.1256$	$p = 1.0$
Percent mean difference in c_{Q2} relative to $RND_{semiactive}$				
1.76	2.32	2.94	1.55	0.0
Significance of mean difference in $\log_{10}(c_{Q2})$ relative to $RND_{semiactive}$				
$p = 0.0594$	$p = 0.0836$	$p = 0.0727$	$p = 0.0769$	$p = 1.0$

Initialisation method				
LEXICO	RND_{active}	$RND_{nondelay}$	*WW*	$RND_{semiactive}$
		$\bar{d}_{lopt\text{-}opt}$		
49.25	64.68	58.55	53.10	70.92
Significance of mean difference in $\bar{d}_{lopt\text{-}opt}$ relative to $RND_{semiactive}$				
$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p < 0.0001$	$p = 1.0$
Percent mean difference in c_{Q2} relative to $RND_{semiactive}$				
1.44	1.07	0.06	−2.79	0.0
Significance of mean difference in $\log_{10}(c_{Q2})$ relative to $RND_{semiactive}$				
$p = 0.5129$	$p = 0.5730$	$p = 0.5555$	$p = 0.3090$	$p = 1.0$

these solutions are typically too far from the nearest optimal solution to have any impact on search cost (e.g., see the right-hand side of Figure 7).

To test this hypothesis, we computed c_{Q2} under each initialisation method (using 1,000 independent trials of $TS_{Taillard}$) for each of the forty-two 10×10 instances with $\leq$100,000 optimal solutions. We then computed the percent difference in c_{Q2} for each method relative to our baseline initialisation method $RND_{semiactive}$; the results are shown in Table 1. We observe a *worst-case* deviation of less than 3%, and the best improvement (obtained under *WW*) is only 2.70%. Further, all observed discrepancies can be attributed to sampling error in the estimates of c_{Q2}, and in no case was the difference in search cost statistically significant (we provide the p-values resulting from a Wilcoxon signed-rank test in Table 1). The data clearly support the hypothesis predicted by our dynamic model: for difficult problems, the choice of initialisation method has no significant impact on the performance of $TS_{Taillard}$.

The results presented in this section concern the impact of initialisation method on the cost required by $TS_{Taillard}$ to locate *optimal* solutions to *difficult* problem instances. Although beyond the scope of this paper, our Markov

model also predicts that initialisation methods can significantly impact the cost of locating both optimal solutions to easy or moderate problem instances and good *sub-optimal* solutions to a wide range of problem instances. We have confirmed these predictions experimentally (Watson, 2003). Finally, we note that our model says nothing about the impact of initialisation method on the performance of tabu search algorithms that employ re-intensification, such as Nowicki and Smutnicki (1996)'s algorithm; we are currently investigating this issue.

6. IMPLICATIONS AND CONCLUSIONS

Our results provide a significant first step toward understanding the dynamics underlying tabu search. We have introduced a dynamic Markov model of Taillard's tabu search algorithm for the JSP. Despite its simplicity, this model accounts for nearly all of the variability in cost required to locate optimal solutions to both small and large random JSPs. The model indicates that Taillard's algorithm can be viewed as a straightforward one-dimensional random walk, with a bias toward solutions that are roughly equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. In contrast to static cost models of problem difficulty, which are based on fitness landscape features, the dynamic model is scalable and provides direct insight into the dynamics of the search process. We also identified a link between static and dynamic models of problem difficulty, providing a hitherto lacking explanation for the success of static models. Finally, we used our dynamic model to gain some insight into the conditions under which different initialisation methods can be expected to improve performance.

Our research also has implications for the design of meta-heuristics. Problem difficulty is dictated by both the size of the local optima sub-space and the strength of the bias toward solutions that are roughly equi-distant from the nearest optimal solution and solutions that are maximally distant from the nearest optimal solution. Given a fixed representation, move operator, and cost function, problem difficulty can be reduced by reducing the strength of this bias. In the limit, when the probabilities of moving both closer to and farther from the nearest optimal solution are 0.5, search cost is known to be polynomial. We conjecture that the most effective meta-heuristics are those that minimise this bias.

Although beyond the scope of the present paper, we have also extended the dynamic model along a number of dimensions (Watson, 2003). First, we have demonstrated that estimation of neighbouring solution makespans, as occurs in Taillard's original algorithm, has no impact on the form or accuracy of the dynamic model. Second, the dynamic model is extensible, in terms of accuracy and to a lesser extent qualitative form, to a variant of Nowicki and Smutnicki's

TSAB algorithm (Nowicki and Smutnicki, 1996) that does *not* perform elite solution recovery (i.e. reintensification); we are currently analysing the impact of elite solution recovery on the dynamic model. Third, the dynamic model also accounts for nearly all of the variability in problem difficulty for more structured JSPs, i.e. those with workflow and flowshop partitions. We have also shown that the qualitative form of the transition probabilities induced by Taillard's algorithm is in fact predictable from the structure of the underlying representation, i.e. the binary hypercube. Another open question regarding our research is generalisation: Do similar results hold for when modelling tabu search algorithms for other *NP*-hard problems? Here, preliminary evidence indicates that similar models can be constructed for tabu search algorithms for the Quadratic Assignment and Permutation Flow-Shop Problems.

Acknowledgments

Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the U.S. Department of Energy under contract DE-AC04-94AL85000. This work was sponsored in part by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-00-1-0144. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

Beck, J. C. and Fox, M. S. (2000) Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, **117**:31–81.

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

Chambers, J. B. and Barnes, J. W. (1996) New tabu search results for the job shop scheduling problem. *Technical Report* ORP96-10, Graduate Programme in Operations Research and Industrial Engineering, The University of Texas at Austin.

Giffler, B. and Thompson, G. L. (1960) Algorithms for solving production scheduling problems. *Operations Research*, **8**:487–503.

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

Jain, A. S., Rangaswamy, B. and Meeran, S. (2000) New and "stronger" job-shop neighborhoods: A focus on the method of Nowicki and Smutnicki (1996) *Journal of Heuristics*, **6**:457–480.

Mattfeld, D. C. (1996) *Evolutionary Search and the Job Shop*. Physica-Verlag, Heidelberg.

Mattfeld, D. C., Bierwirth, C. and Kopfer, H. (1999) A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, **86**:441–453.

Nowicki, E. and Smutnicki, C. (1996) A fast taboo search algorithm for the job shop problem. *Management Science*, **42**:797–813.

Singer, J., Gent, I. P. and Smaill, A. (2000) Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, **12**:235–270.

Taillard, É. D. (1989) Parallel taboo search technique for the jobshop scheduling problem. *Technical Report* ORWP 89/11, DMA, Ecole Polytechnique Fédérale de Lausanne, Switzerland.

Taillard, É. D. (1994) Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, **6**:108–117.

van Laarhoven, P. J. M, Aarts, E. H. L. and Lenstra, J. K. (1992) Job shop scheduling by simulated annealing. *Operations Research*, **40**:113–125.

Watson, J. P. (2003) *Empirical modeling and analysis of local search algorithms for the jobshop scheduling problem. Ph.D. Thesis*, Department of Computer Science, Colorado State University.

Watson, J. P., Beck, J. C., Howe, A. E. and Whitley, L. D. (2003) Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, **143**:187–217.

Werner, F. and Winkler, A. (1995) Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, **58**:191–211.

Bin Packing

THE BEST-FIT RULE FOR MULTIBIN PACKING: AN EXTENSION OF GRAHAM'S LIST ALGORITHMS

Pierre Lemaire, Gerd Finke and Nadia Brauner
Laboratoire Leibniz-IMAG
46 avenue Felix Viallet, 38031 Grenoble cedex, France
{ pierre.lemaire, gerd.finke, nadia.brauner } @imag.fr

Abstract In this paper, we deal with multibin packing problems. These problems are linked to multiprocessor-task scheduling as well as to bin packing problems: they consist of n objects to be packed into m bins, with each object requiring space in several bins.

We propose an intuitive greedy approach (the best-fit rule), which extends the well-known list algorithms for multiprocessor scheduling, to solve the case when objects have fixed height and size. We prove that it provides a 2-approximation, and even a 4/3-approximation if the objects are sorted by non-increasing heights. Based on this method, a polynomial time approximation scheme (PTAS) will be developed.

Keywords: multibin packing, approximation algorithms, list algorithms, PTAS.

1. MULTIBIN PACKING

The concept of *multibin packing*, presented in Lemaire *et al.* (2003), is closely linked to the well-known problems of bin packing and multiprocessor-task scheduling. In a multibin problem, there are n objects to be packed into m bins, and each object occupies a certain height in several bins. Different models and objectives exist: Lemaire *et al.* (2003) propose a classification for multibin packing problems, based on the one that exists for multiprocessor-task scheduling (see Błażewicz *et al.*, 1994; Drozdowski, 1996).

Actually, multibin packing problems are very close to multiprocessor-task scheduling problems. Indeed, the former is a relaxation of the latter: the constraint that the different parts of a task are performed in parallel is dropped. Hence, to get a multibin packing, one can turn by 90° the Gantt chart of a multiprocessor-task scheduling problem, and then let the tasks drop (see Figure 1).

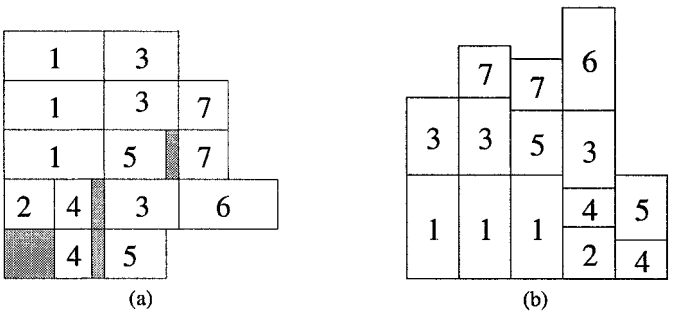


Figure 1. (a) A multiprocessor-task schedule; (b) the corresponding multibin packing.

Multibin packing problems are not just a mere relaxation of multiprocessor-task scheduling problems. They also include several well-known problems such as parallel machine scheduling (Chen *et al.*, 1998), bin packing (Coffman Jr *et al.*, 1997), bipartition, knapsack problems (Kellerer *et al.*, 2004), and ring loading problems (Schrijver *et al.*, 1998). Above all they have a wide range of applications of their own, for instance document analysis, telecommunication network design, project building, drug testing (Lemaire *et al.*, 2003). Typically, whenever some tasks have to be performed by several operators without any synchronisation requirement, it is a multibin object packing problem. To give a better feeling on what kind of problem we deal with, the document analysis problem is describe with more details below. Before this, let us formalise the model we are interested in.

In this paper, we focus on the so-called $B|size_j|H_{\max}$ problems. There are n objects. Each object O_j is defined by its height h_j and its width (number of parts) $size_j$. The $size_j$ parts of an object O_j must be packed into distinct bins, where they occupy the height h_j. Let $h_{\max}$ be the height of the highest object: $h_{\max} = \max_{j=1..n} h_j$.

Consider a feasible multibin packing with m identical bins. Let the height H_i of a bin B_i be the sum of the heights of the objects packed into it. The order of the objects within a bin is of no importance. Let $H_{\max}$ be the height of the highest bin: $H_{\max} = \max_{i=1..m} H_i$.

The objective is to minimise $H_{\max}$; we call the optimal value $H^*_{\max}$.

To give an illustration, let us describe the document analysis problem. A group of experts have to examine documents, and each document must be examined by several experts according to its importance. A certain time is allocated for analysing each of the documents. The objective is to expertise all documents as quickly as possible. In this problem, the experts correspond to

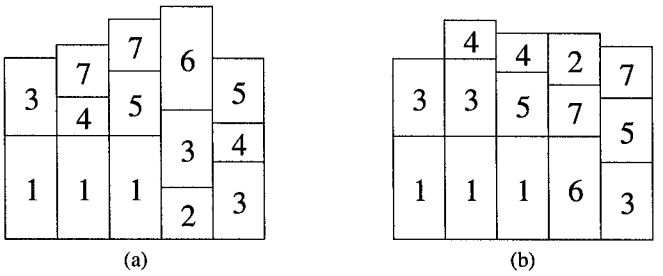


Figure 2. Applying the BF rule to the example of Figure 1; (a) natural order; (b) non-increasing height order.

the bins, the documents to the objects and the objective is to minimise the total height packed into the highest bin.

$B|size_j|H_{\max}$ problems are strongly $\mathcal{NP}$-hard in the general case, and weakly $\mathcal{NP}$-hard if the number of bins m is fixed (i.e. case $Bm|size_j|H_{\max}$). For more details, in particular concerning a pseudo-polynomial algorithm to solve this last case, we refer the reader to Lemaire *et al.* (2003).

The remainder of this paper deals with a greedy principle that we call the "best-fit rule", which can be seen as a generalisation of the list algorithms by Graham (1969): in Section 2, we introduce the best-fit rule and we use it to design some very efficient algorithms with good performance guarantees. In Section 3, we extend these results to derive a PTAS (Polynomial Time Approximation Scheme) for $Bm|size_j|H_{\max}$.

2. THE BEST-FIT RULE

In this section, a simple packing rule called the Best-Fit rule is described and its properties are analysed.

The *Best-Fit (BF) rule* may be seen as an adaptation to multibin packing of the well-known "list scheduling" for multiprocessor scheduling presented in Graham (1969). The BF rule is to always pack the $size_j$ parts of an object O_j into the $size_j$ lowest bins[1], the height of a bin being the sum of the heights of the objects packed into it (see Figure 2(a)). Note that the solution computed according to the BF rule strongly depends on the order on the objects (Figure 2(b)). In particular sorting the objects by non-increasing height leads to better guarantees, as proved in Section 2.2.

[1]Our "best-fit" rule is very similar to the "worst-fit" rule of bin packing. However, there is no contradiction and the name is reasonable since we consider the dual objective.

A BF algorithm may be implemented in time $\mathcal{O}(nm)$ (there is no need to sort all bins at each iteration). Hence, one has a polynomial-time approximation algorithm for the problem $Bm|size_j|H_{\max}$, but only a pseudo-polynomial approximation algorithm for the general case $B|size_j|H_{\max}$. For practical purposes, however, it is very fast and efficient.

2.1 Properties of the BF Rule

The BF rule provides several guarantees on the solutions built, as shown by the following properties. Note that most of these properties do not require that a solution is entirely built according to the BF rule: some initial assignment may be fixed for some objects, and so we introduce the "initial difference of height between bins i and j", i.e. the difference of height $\Delta_{ij}(0)$ that exists because of pre-assignments and before the BF phase starts. Similarly we define the "initial difference of height" $\Delta(0)$ which is the maximal initial difference of height between two bins.

Property 1 *Let n objects be packed according to the BF rule. Let $\Delta_{ij}(k)$ be the height-difference between bins B_i and B_j after the kth object, of height h_k, is packed. Then,*

$$\forall k = 1, 2, \ldots, n : \Delta_{ij}(k) \leq \max(\Delta_{ij}(k-1), h_k - \Delta_{ij}(k-1))$$

with $\Delta_{ij}(0)$ being the initial difference of height before applying the BF rule.

Proof. Let $H_i(k)$ be the height of bin i after object O_k is packed. Because of the definition of $\Delta_{ij}(0)$, the property is initially true. Suppose now that it is true for $k - 1 < n$. If O_k is neither packed into B_i nor B_j, then $\Delta_{ij}(k) = \Delta_{ij}(k-1)$. If it is packed into both then, again, $\Delta_{ij}(k) = \Delta_{ij}(k-1)$. If it is only packed into one bin, then it must be the smallest one (say B_i). In this last case: if $H_j(k) \geq H_i(k)$ then $\Delta_{ij}(k) = \Delta_{ij}(k-1) - h_k \leq \Delta_{ij}(k-1)$; otherwise $\Delta_{ij}(k) = h_k - \Delta_{ij}(k-1)$. □

A direct consequence of this property is the following.

Property 2 *With the same assumptions as for Property 1, let $h_{\max}$ be the greatest height of an object packed according to the BF rule. Then,*

(a) if $\Delta_{ij}(k) \leq h_{\max}$ then $\forall k', k \leq k' \leq n : \Delta_{ij}(k') \leq h_{\max}$

(b) if $\Delta_{ij}(n) \geq h_{\max}$ then $\forall k, k \leq n : \Delta_{ij}(k) \geq h_{\max}$.

A similar result as for Δ_{ij} is obtained for Δ, the maximal difference of heights between two bins.

Property 3 *Let n objects be packed according to the BF rule. Let $\Delta(k)$ be the greatest height-difference between any two bins after the kth object, of height h_k, is packed. Then,*

$$\forall k = 1, 2, \ldots n : \Delta(k) \leq \max(\Delta(k-1), h_k)$$

with $\Delta(0)$ being the initial greatest difference of heights before applying the BF rule.

Proof. This is a direct consequence of Property 1. We use the same notation. For any k, there are two bins B_a, B_b such that $\Delta(k) = H_a(k) - H_b(k) = \Delta_{ab}(k)$. We obtain

$$\begin{aligned} \Delta(k) &= \Delta_{ab}(k) \\ &\leq \max_{ij}\{\Delta_{ij}(k-1), h_k - \Delta_{ij}(k-1)\} \\ &\leq \max(\max_{ij}\{\Delta_{ij}(k-1)\}, h_k) \\ &\leq \max(\Delta(k-1), h_k) \end{aligned}$$

which is the claim. □

These properties, together with the following lemma, yield guarantees on the quality of a solution computed by the BF rule.

Lemma 1 *Let S be a feasible solution for instance I of value $H_{\max}(S)$. Then,*

$$H_{\max}(S) \leq H^*_{\max} + \frac{m-1}{m}\Delta$$

where Δ is the greatest difference of heights between two bins of S.

Proof. There is in S at least one bin of height $H_{\max}(S)$. Then $m-1$ bins are of height at least $H_{\max}(S) - \Delta$. As every object is packed at most (indeed exactly) once,

$$\sum_{j=1}^{n} h_j.size_j \geq H_{\max}(S) + (m-1)(H_{\max}(S) - \Delta) = mH_{\max}(S) - (m-1)\Delta$$

Furthermore, the following clearly holds:

$$H^*_{\max} \geq \frac{\sum_{i=1}^{n} h_j.size_j}{m}$$

By combining these two inequalities, we obtain

$$H^*_{\max} \geq H_{\max}(S) - \frac{m-1}{m}\Delta$$

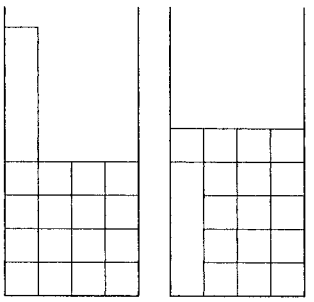


Figure 3. An example where the bound of Theorem 4 is tight. (a) Worst case for BF; (b) optimal solution.

which is the claim. □

Theorem 4 *Let S be a solution obtained from a BF algorithm. Then,*

$$H_{\max}(S) \leq H^*_{\max} + \frac{m-1}{m}\Delta \qquad \text{with} \qquad \Delta \leq \max(h_{\max}, \Delta(0))$$

and this bound is tight.

Proof. From Property 3, induction leads to $\Delta(j) \leq \max(h_{\max}, \Delta(0))$ for all j. In particular, $\Delta = \Delta(n) \leq \max(h_{\max}, \Delta(0))$. Applying Lemma 1 completes the proof. □

The bound given in Theorem 4 is tight: we cannot expect a BF algorithm to be more efficient than the above guarantee (unless some further assumptions are made on the order of the objects), even with objects of size 1. For example, consider the following instance: m bins and $m^2 + 1$ objects of size 1 with heights: $h_1 = h_2 = \cdots = h_{m^2} = 1$ and $h_{m^2+1} = m$. On such an instance, a BF algorithm builds a solution S of value $H_{\max}(S) = 2m = H^*_{\max} + \frac{m-1}{m}h_{\max}$, if the objects are considered in the order they are defined (see Figure 3).

Corollary 1 *A BF algorithm is asymptotically optimal for all uniformly bounded instances of $Bm|size_j|H_{\max}$ (i.e. there are uniform bounds $h_{\max} \geq h_{\min} > 0$ such that for any object O_j, then $h_{\max} \geq h_j \geq h_{\min}$).*

Proof. The upper bound on h_j guarantees that Δ is bounded, whereas the lower bound on h_j guarantees that $H^*_{\max}$ is unbounded (the h_j cannot be the terms of a convergent sequence) when we add objects. That is,

$$1 \leq \frac{H_{\max}(S)}{H^*_{\max}} \leq 1 + \frac{\Delta}{H^*_{\max}} \qquad \text{and} \qquad \frac{\Delta}{H^*_{\max}} \rightsquigarrow_{n\to\infty} 0$$

□

This corollary has some practical importance: it implies that the larger the instance is, the better BF algorithms are. Some tests on randomly generated instances confirm this. Moreover, as $h_{\max} \leq H^*_{\max}$, we get a multiplicative performance ratio for BF algorithms.

Corollary 2 *Let S be a solution completely*[2] *obtained from a BF algorithm. Then,*

$$H_{\max}(S) \leq (2 - \frac{1}{m})H^*_{\max} < 2H^*_{\max}$$

and this bound is tight.

This bound is indeed the same as the one proved in Graham (1969), for the special case of objects of size one. Moreover, Graham shows that this bound is tight for his special case, so it is tight in our case too. For the special case of equal-height objects, we have the following corollary.

Corollary 3 *BF algorithms are optimal for $B|size_j, h_j = 1|H_{\max}$.*

Proof. If $h_{\max} = 1$, then Theorem 4 implies that $H^{BF}_{\max} < H^*_{\max} + 1$. Because of the integrality of the values, we have $H^{BF}_{\max} = H^*_{\max}$. □

A consequence is that BF algorithms are polynomial time algorithms to solve $Bm|size_j, h_j = 1|H_{\max}$ but only pseudo-polynomial time algorithms for $B|size_j, h_j = 1|H_{\max}$ (i.e. if the number of bins is not fixed). However Lemaire *et al.* (2003) propose an algorithm—which is essentially a special implementation of the BF rule—that solves this latter case in polynomial time.

Theorem 4 can also be extended to the case where some objects have a pre-assignment.

Theorem 5 *Let I be an instance of $B|size_j|H_{\max}$. Let $\mathbf{O}_1, \mathbf{O}_2$ be a partition of its set of objects $\mathbf{O}$. Let $S(\mathbf{O}|\mathbf{O}_1)$ be a solution for all objects $\mathbf{O}$, given an assignment $S(\mathbf{O}_1)$ of $\mathbf{O}_1$, and let $H^*_{\max}(\mathbf{O}|\mathbf{O}_1)$ be the value of an optimal solution with regard to this assignment. Complete $S(\mathbf{O}_1)$ with $\mathbf{O}_2$ according to the BF rule to build a solution S for I. We have*

$$(a) \qquad H_{\max}(S) \leq H^*_{\max} + \frac{m-1}{m} \max(h_{\max}(\mathbf{O}_2), \Delta(\mathbf{O}_1))$$

and

$$(b) \qquad H_{\max}(S) \leq H^*_{\max}(\mathbf{O}|\mathbf{O}_1) + \frac{(m-1)^2}{m} h_{\max}(\mathbf{O}_2)$$

where $h_{\max}(\mathbf{O}_2) = \max\{h_j, O_j \in \mathbf{O}_2\}$,
and $\Delta(\mathbf{O}_1) = \max\{|H_i(S(\mathbf{O}|\mathbf{O}_1)) - H_j(S(\mathbf{O}|\mathbf{O}_1))|, 1 \leq i < j \leq m\}$.

[2] S is "completely" obtained from a BF algorithm means that $\Delta(0) = 0$.

Proof. Since the order of the objects within the bins is of no importance, we can assume without any loss of generality that for all solutions, the objects of $\mathbf{O}_1$ are at the bottom of the bins.

Part (a) is indeed a reformulation of Theorem 4. We now turn to the (b) part. The proof is done by induction on the number m of bins. Our induction hypothesis is that Theorem 5(b) is true for every instance with $k < m$ bins. We have to prove that this implies that Theorem 5(b) is true for any instance with m bins.

In the case $m = 1$, there is no instance with $k < m$ bins. Hence the induction hypothesis is true. If $m = 2$, every solution for one bin is optimal and the induction hypothesis is true too.

Let I be an arbitrary instance defined on m bins.

First, suppose that, at the end of the procedure, the maximal difference of heights verifies: $\Delta \leq (m-1)h_{\max}(\mathbf{O}_2)$. In this case, we do not even require the induction hypothesis: by Lemma 1, we know that

$$H_{\max}(S) \leq H^*_{\max} + \frac{m-1}{m}(m-1)h_{\max}(\mathbf{O}_2) = H^*_{\max} + \frac{(m-1)^2}{m}h_{\max}(\mathbf{O}_2)$$

with $H^*_{\max}$ being the value of an optimal solution of I, and thus $H^*_{\max} \leq H^*_{\max}(\mathbf{O}|\mathbf{O}_1)$.

Now, suppose that $\Delta > (m-1)h_{\max}(\mathbf{O}_2)$. Let us normalise the solution S so that the bins $B_1, B_2, \ldots, B_m$ are sorted in non-increasing height at the end of the procedure.

Since $\Delta > (m-1)h_{\max}(\mathbf{O}_2)$, there must be an index i such that $H_i > H_{i+1} + h_{\max}(\mathbf{O}_2)$. Property 2$(b)$ implies that the height H_j for any $B_j, j \geq i+1$ satisfies $H_j < H_k$ for all $B_k, k \leq i$ at each step of the BF phase. So, according to the BF rule, any object of $\mathbf{O}_2$ put into a bin $B_k, k \leq i$ has also been packed into all bins $B_j, j > i$.

Therefore the solution S satisfies the following: every object O_j with $size_j \leq m - i$ is completely packed into bins $B_{i+1}, B_{i+2}, \ldots, B_m$; and every object O_j with $size_j > m - i$ occupies all bins $B_{i+1}, B_{i+2}, \ldots, B_m$ and $size_j - (m-i)$ of the bins $B_1, B_2, \ldots, B_i$. This means that for all solutions with the same assignment for $\mathbf{O}_1$, one cannot put less volume into bins $B_1, B_2, \ldots, B_i$.

Therefore, we can reduce the problem to the first i bins, $B_1, B_2, \ldots, B_i$. The set $\mathbf{O}_2$ is reduced to objects that have $size_j > m - i$, replacing $size_j$ by $size^R_j = size_j - (m-i)$. The solution S is reduced to S^R by keeping only the first i bins (for this reduced solution, the fixed part is $\mathbf{O}^R_1$, and $\mathbf{O}^R_2$ is packed according to the BF rule).

Since $i < m$, we can use the induction hypothesis on the reduced problem

$$H_{\max}(S^R) \leq H^*_{\max}(\mathbf{O}^R|\mathbf{O}^R_1) + \frac{(i-1)^2}{i}h_{\max}(\mathbf{O}^R_2)$$

We also know that $H_{\max}(S^R) = H_1 = H_{\max}(S)$ and $H^*_{\max}(\mathbf{O}^R|\mathbf{O}^R_1) \leq H^*_{\max}(\mathbf{O}|\mathbf{O}_1)$ since we cannot put less into bins $B_1, B_2, \ldots B_i$, as explained above. Thus,

$$H_{\max}(S) = H_{\max}(S^R) \leq H^*_{\max}(\mathbf{O}^R|\mathbf{O}^R_1) + \frac{(i-1)^2}{i} h_{\max}(\mathbf{O}^R_2)$$

and as $\frac{(i-1)^2}{i} < \frac{(m-1)^2}{m}$ and $h_{\max}(\mathbf{O}^R_2) \leq h_{\max}(\mathbf{O}_2)$, we obtain

$$H_{\max}(S) \leq H^*_{\max}(\mathbf{O}|\mathbf{O}_1) + \frac{(m-1)^2}{m} h_{\max}(\mathbf{O}_2)$$

This concludes the proof. □

The natural question that arises from the previous theorem is: What if the assignment used for $\mathbf{O}_1$ is optimal? To answer this question, we adopt a different approach based on reducing solutions of m bins to solutions of $m-1$ bins.

Definition 1 *Let I be an instance of $B|size_j|H_{\max}$ with m bins. Let S be a solution for I. We define a* reduction *R of S by $B_k, k \in \{1, 2, \ldots, m\}$ as S with the bin B_k removed (with everything it contains). We define I^R as I with everything in B_k removed, and $m-1$ bins. If $\mathbf{O}$ is the set of objects of I, we note $\mathbf{O}^R$ the set of objects of I^R.*

Note that R is a valid solution for I^R. Furthermore, reducing solutions preserves optimality and BF construction, as shown by the two following lemmas.

Lemma 2 *Let I be an instance of $B|size_j|H_{\max}$ with m bins. Let S^* be an optimal solution for I. Let $B_k \in S^*$ such that $H_k < H^{S^*}_{\max}$ and let R be a reduction of S^* by B_k. Then,*

(a) R is an optimal solution for I^R,

(b) if S' is an optimal solution for I^R, then $S' \cup B_k$ is an optimal solution of I,

(c) I and I^R share the same optimal value.

Proof. Statement (a). If S^R is not optimal then consider an optimal solution for I^R and add the removed objects to bin B_k : one obtains, in this way, a solution for I which is strictly better than S^*, which is impossible.

Statements (b) and (c). By statement (a), S^R is optimal for I^R, and thus has the same value as S'. As a consequence, $H_k < H^{S'}_{\max}$ and thus $S' \cup B_k$ is a feasible solution for I, of the same value as S^*. As a consequence, $S' \cup B_k$ is

optimal, and S' and S^* have the same value (that is, I and I^R share the same optimal value). □

Lemma 3 *Let S be a solution computed by BF with regard to a given order. Let R be a reduction of S by a certain bin B_k. Then,*

$$R = BF(I^R)$$

if $BF(I^R)$ is computed with regard to the same order. This equality means that both solutions have not only the same value, but also that objects are packed in the same bins (up to equal objects).

Proof. This is proved recursively: the first part (not in B_k) of the first object is placed in the lowest bin, which is the same in R and $BF(I^R)$.

Now suppose every part so far had been placed in the same bin in R and $BF(I^R)$. Then consider the next part p of a certain object O_j.

If, in S, p was packed in B_k, then it had been removed from I^R.

If, in S, p was packed in bin $B_i \neq B_k$, i.e. B_i was the lowest bin without any part of the same object. The induction assumption implies that the same parts are packed in equivalent places in R and $BF(I^R)$, thus p is packed in B_i for solution $BF(I^R)$ too.

As a conclusion, R and $BF(I^R)$ are the same solution (except for equal objects, of which some parts may have been exchanged). □

Theorem 6 *Let I be an instance of $B|size_j|H_{\max}$. Let $\mathbf{O}_1, \mathbf{O}_2$ be a partition of its set of objects $\mathbf{O}$. Let S^* be an optimal solution for all objects $\mathbf{O}$ and let $H^*_{\max}$ be its value. Let $S^*(\mathbf{O}_1)$ be an optimal assignment of $\mathbf{O}_1$ and let S be a solution built by completing $S^*(\mathbf{O}_1)$ with $\mathbf{O}_2$ according to the BF rule. We have*

$$H_{\max}(S) \leq H^*_{\max} + h_{\max}(\mathbf{O}_2) \quad \text{with} \quad h_{\max}(\mathbf{O}_2) = \max\{h_j, O_j \in \mathbf{O}_2\}$$

and this bound is tight, even for unit-height objects.

Proof. The proof is done recursively on the number of bins m: we prove that, if there is no instance with m bins such that $H_{\max}(S) > H^*_{\max} + h_{\max}(\mathbf{O}_2)$, for a certain partition $\mathbf{O} = \mathbf{O}_1 \cup \mathbf{O}_2$, then there is no instance with $m+1$ bins and the given property either.

To initiate the recurrence, note that, if there is only 1 bin, every solution is optimal and thus there is no solution such that $H_{\max}(S) > H^*_{\max} + h_{\max}(\mathbf{O}_2)$.

Now let us suppose there exists S, a solution with $m+1$ bins built as described in the theorem and such that $H_{\max}(S) > H^*_{\max} + h_{\max}(\mathbf{O}_2)$. We

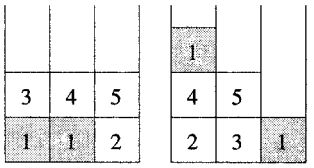


Figure 4. An example where the bound of Theorem 6 is tight. (a) An optimal solution; (b) a solution computed according to Theorem 6, with $\mathbf{O}_1 = \{O_2, O_3, O_4, O_5\}$ and $\mathbf{O}_1 = \{O_1\}$.

shall prove that, if such a S exists, then it exists a similar S for m bins, which contradicts the induction assumption.

Let B_k be a bin of S such that $H_k < H_{\max}(S)$. B_k exists: otherwise, S would be optimal. Let R be the reduction of S by B_k.

We can apply Lemma 2 and $R(\mathbf{O}_1)$ is an optimal assignment of $\mathbf{O}_1^R$. Furthermore, by Lemma 3, the BF phase packed the objects in the same places. Thus, R is built as S: an optimal assignment of $\mathbf{O}_1^R$, then a BF procedure for $\mathbf{O}_2^R$. Furthermore,

$$\begin{array}{lcll} H_{\max}(R) & = & H_{\max}(S) & (B_k < H_{\max}(S)) \\ & > & H^*_{\max}(I) + h_{\max}(\mathbf{O}_2) & (S \text{ is a counter-example}) \\ & \geq & H^*_{\max}(I^R) + h_{\max}(\mathbf{O}_2^R) & (H^*_{\max}(I) = H^*_{\max}(I^R) \text{ [lemma 2(c)]}; \\ & & & h_{\max}(\mathbf{O}_2) \geq h_{\max}(\mathbf{O}_2^R)) \end{array}$$

Altogether, if S is a counter-example for I on $m+1$ bins, then R is a counter-example for I^R on m bins. Hence, we have the announced contradiction, which proves the theorem.

To prove that this bound is tight even in the case of unit-height objects, consider the following instance. There are m bins, and $n = 2m - 1$ objects of height $h_j = 1, \forall j$. Furthermore: $size_1 = 2$, and $size_j = 1, \forall j \geq 2$. As shown in Figure 4, the optimal solution has value 2, whereas a solution computed according to Theorem 6 may have value $3 = 2 + h_{\max}(\mathbf{O}_2)$, with $\mathbf{O}_2 = \{O_1\}$ and $\mathbf{O}_2 = \mathbf{O} \backslash \mathbf{O}_2$. □

Notice that this last theorem does not require any of the previously proved properties on BF solutions. Furthermore, it also implies that the BF rule has the guarantee of not exceeding the optimal solution by more than $h_{\max}$, and thus that it is a 2-approximation. We can add the following corollary.

Corollary 4 *Let S be a solution built by packing optimally the k highest objects and then by packing the remaining objects according to the BF rule. Such a S satisfies*

$$H_{\max}(S) \leq \left(1 + \frac{1}{1 + \left\lfloor \frac{k}{m} \right\rfloor}\right) H^*_{\max}$$

Proof. Let $\mathbf{O}_1$ be the set of the k highest objects, and let $\mathbf{O}_2$ be the set of all remaining objects. Let $h_{\max}(\mathbf{O}_2)$ be the highest height of an object of $\mathbf{O}_2$.

There are at least $k+1$ objects of height $h_{\max}(\mathbf{O}_2)$ or more. Thus, at least one bin contains at least $1+\lfloor k/m \rfloor$ such objects. Therefore, $h_{\max}(\mathbf{O}_2)$ satisfies

$$H^*_{\max} \geq (1 + \lfloor k/m \rfloor) h_{\max}(\mathbf{O}_2)$$

that is

$$h_{\max}(\mathbf{O}_2) \leq \frac{1}{1 + \lfloor \frac{k}{m} \rfloor} H^*_{\max}$$

By Theorem 6, the solution S satisfies

$$H_{\max}(S) \leq H^*_{\max} + h_{\max}(\mathbf{O}_2)$$

that is

$$H_{\max}(S) \leq (1 + \frac{1}{1 + \lfloor \frac{k}{m} \rfloor}) H^*_{\max}$$

□

2.2 The Decreasing BF Case

So far, no assumption on the order of the objects was made. We turn now to a particular case. We call a BF algorithm a Decreasing Best-Fit (DBF) algorithm if it considers the objects by non-increasing height, i.e. $h_1 \geq h_2 \geq \cdots \geq h_n$.

Lemma 4 *DBF is optimal if any of the following holds:*
(a) $\forall O_j \in \mathbf{O} : h_j > H^*_{\max}/3$*;*
(b) at most two objects are packed in the highest bin of a DBF solution;
(c) there is an optimal solution with at most two objects packed in any bin.

Proof. It is clear that (a) follows from (c), so we shall prove (c). The technique was already used in Graham (1969) for a similar result. Consider an optimal packing, with at most 2 objects per bin. We shall transform it into a DBF packing of same value.

Step 1: as long as there are two bins B_j and B_k with two objects O_a and O_b in B_j and just one object O_c in B_k such that: $h_a > h_c$ and $O_b \neq O_c$, pack O_b and O_c in B_j and pack O_a in B_k. This transformation is always valid, and does not worsen the solution.

Step 2: as long as there are two bins B_j and B_k with objects O_a and O_b in B_j and objects O_c and O_d in B_k such that: $h_a > h_c$, $h_b > h_d$, $O_a \neq O_d$ and $O_b \neq O_c$, pack O_b and O_c in B_j and pack O_a and O_d in B_k. This transformation is always valid, and does not worsen the solution.

Step 3: sort the bins by non-increasing height of their highest object.

Eventually, we get a normal form of an optimal solution, which indeed is a DBF solution.

Proof of (b). If there is just one object in the highest bin of a solution, then this solution is optimal. Suppose that there are two objects packed into the highest bin (say, B_i) of a DBF solution. Let O_a be the first one packed and O_b the second one. Since two objects are packed together, then there is no empty bin left and, because of the BF rule, O_a is the smallest object that can be packed with O_b. Because of the order on the objects, O_b is the smallest object packed so far. There may be objects between O_a and O_b but those objects must be packed by two (otherwise, O_a would not be chosen by the BF rule), and they are at least as big as O_b (thus, exchanging them with O_b would worsen the solution). Hence, any solution where O_a and O_b are not packed together is worse than the DBF solution: this latter solution is optimal. □

Remark 7 *Assumption (c) cannot be weakened to: "there is an optimal solution with at most two objects packed in its highest bin" (e.g. consider the instance: $m = 2$, $n = 6$, $\forall j : size_j = 1$ and $h_1 = 5, h_2 = 3, h_3 = h_4 = h_5 = h_6 = 2$).*
Assumption (b) is not implied by $\sum_{O_j \in \mathbf{O}} size_j \leq 2m$ (e.g. consider the instance: $m = 2$, $n = 4$, $\forall j : size_j = 1$ and $h_1 = 5, h_2 = h_3 = h_4 = 2$). Indeed, in this latter case, DBF is not always optimal (e.g. consider the instance: $m = 4$, $n = 8$, $\forall j : size_j = 1$ and $h_1 = h_2 = 8, h_3 = 5, h_4 = 3, h_5 = h_6 = h_7 = h_8 = 2$).

Theorem 8 *Decreasing Best-Fit (DBF) is a $4/3$-approximation.*

Proof. Let us define $\mathbf{O}_1 = \{O_j \in \mathbf{O}, h_j > H^*_{\max}/3\}$ and $\mathbf{O}_2 = \{O_j \in \mathbf{O}, h_j \leq H^*_{\max}/3\}$. DBF packs first the objects of $\mathbf{O}_1$. By lemma 4(a), these objects are packed optimally. Thus, by Theorem 6, we have the guarantee that

$$\begin{aligned} H^{DBF}_{\max} &\leq H^*_{\max} + h_{\max}(\mathbf{O}_2) \\ &\leq H^*_{\max} + \frac{H^*_{\max}}{3} \\ &\leq \frac{4}{3} H^*_{\max} \end{aligned}$$

□

2.3 Best-Fit Algorithms vs Graham's List Algorithms

The bounds of Theorems 4, 6, and 8 generalise the well-known bounds by Graham (1969) and Coffman and Sethi (1976) for $P||C_{\max}$ (i.e. $B|size_j = 1|H_{\max}$). When applied to $B|size_j = 1|H_{\max}$,

(a) BF algorithms are $(2 - \frac{1}{m})$-approximation algorithms;

(b) DBF algorithms are $(\frac{4}{3} - \frac{1}{3m})$-approximation algorithms;

(c) to pack the k highest objects optimally, and then to pack the other objects by a BF algorithm is a $\left(1 + \frac{1-\frac{1}{m}}{1+\lfloor \frac{k}{m} \rfloor}\right)$-approximation algorithm;

(d) DBF algorithms are exact if $r = 1, 2$ and $(1 + \frac{1}{r} - \frac{1}{rm})$-approximation algorithms otherwise (with r being the number of objects packed into a bin of maximal height).

Parts (a)–(c) were proved in Graham (1969), and the (d) part is due to Coffman and Sethi (1976). The (c) part includes both (a) and (b) as special cases, but those are the best-known bounds: (a) is the general bound for list-scheduling algorithms, whereas (b) is the bound for LPT (Longest Processing Time first) algorithms.

In our case, with no restriction on the size of the objects, the bound (a) still holds (Corollary 2).

The bound (b) is weakened: if the objects are all of size 1, Graham proves a guarantee of $\frac{4}{3} - \frac{1}{3m}$ whereas we have a guarantee of $\frac{4}{3}$ for DBF (Theorem 8). It is an open question whether this corrective term $(-\frac{1}{3m})$ is also valid in our multibin case. However, for $m = 2$, DBF meets the $\frac{4}{3} - \frac{1}{3m}$ performance ratio; furthermore, since Graham's bound is tight, one can get as close to our bound as one wants, as m grows.

With Theorem 6, we generalise both the problem (the objects are not required to be of size 1) and the assumptions on $\mathbf{O}_1$ (which is not necessary the set of the highest objects), with regard to the (c) part. In the process, as for the DBF case, we lose a corrective term when going back to the case when $\mathbf{O}_1$ is the set of the k highest objects: Graham proved a performance of $1 + \frac{1-1/m}{1+\lfloor k/m \rfloor}$ whereas we have $1 + \frac{1}{1+\lfloor k/m \rfloor}$ (Corollary 4).

The (d) part is a possible further extension. Note that we have already proved that DBF are exact if r is 1 or 2 (Lemma 4).

3. A PTAS FOR MULTIBIN PACKING

In this section we describe a PTAS (Polynomial Time Approximation Scheme) for the problem $Bm|size_j|H_{\max}$, that is an algorithm which is a $(1 + \epsilon)$-approximation and which runs in polynomial time for any fixed m and ϵ.

A classical approach for the design of a PTAS is to split the set of objects into "big" ones and "small" ones. Then, exhaustive search is performed on the big objects (that can be done if they are not too numerous) whereas small objects are coped with using a fast heuristic with performance guarantee (which is good if the objects are small enough). The key is to find a good compromise so as to actually have neither too many big objects, nor small objects that are

too large. Such a method has already been successfully applied to related problems (see Khanna, 1997; Chen and Miranda, 2001). In our case, we propose the following scheme.

Consider an instance of $Bm|size_j|H_{\max}$, and let β be a positive real number. We split the set $\mathbf{O}$ of objects into (respectively) the big and the small objects as follows: $\mathbf{O}_{big} = \{O_j \in \mathbf{O}, h_j > \beta\}$ and $\mathbf{O}_{small} = \{O_j \in \mathbf{O}, h_j \leq \beta\}$. Then, for an optimal assignment of the objects in $\mathbf{O}_{big}$, we complete the solution with a DBF algorithm on the small objects.

We now claim that, for a correct choice of β, this leads to a PTAS for $Bm|size_j|H_{\max}$ (see Algorithm 1).

Algorithm 1: (instance I, positive real ϵ)

1. compute $H_{approx} = DBF(I)$.
2. compute $\beta = \epsilon\frac{3}{4}H_{approx}$.
3. partition the object set $\mathbf{O}$ into:
   $\mathbf{O}_{big} = \{O_j \in \mathbf{O}, h_j > \beta\}$
   $\mathbf{O}_{small} = \{O_j \in \mathbf{O}, h_j \leq \beta\}$
4. find an optimal assignment for $\mathbf{O}_{big}$.
5. complete this assignment with $\mathbf{O}_{small}$ by DBF

Theorem 9 *Algorithm 1 is a PTAS for $Bm|size_j|H_{\max}$ that runs in polynomial time $\mathcal{O}(n\ln(n) + nm + 2^{\frac{4m^2}{3\epsilon}})$.*

Proof. In step 1, H_{approx} is computed by a DBF algorithm. By Theorem 8,

$$H_{approx} \leq \frac{4}{3}H^*_{\max}$$

Hence, the highest object of $\mathbf{O}_{small}$ verifies

$$h_{\max}(\mathbf{O}_{small}) \leq \beta \leq \epsilon\frac{3}{4}H_{approx} \leq \epsilon H^*_{\max}$$

By Theorem 6, a solution S computed by Algorithm 1 verifies

$$H_{\max}(S) \leq H^*_{\max} + h_{\max}(\mathbf{O}_{small}) \leq (1+\epsilon)H^*_{\max}$$

Hence, Algorithm 1 is a $(1+\epsilon)$-approximation.

The running time of step 1 is $\mathcal{O}(n\ln(n) + nm)$; step 2 is $\mathcal{O}(1)$; step 3 is $\mathcal{O}(n)$; step 5 is $\mathcal{O}(nm)$. For step 4, an optimal assignment can be found by exhaustive search. Let K be the number of possible assignments of $\mathbf{O}_{big}$:

$$K \leq \prod_{O_j \in \mathbf{O}_{big}} \binom{m}{size_j} \leq \prod_{O_j \in \mathbf{O}_{big}} 2^m \leq 2^{m|\mathbf{O}_{big}|}$$

Note that

$$|\mathbf{O}_{big}| \leq \sum_{O_j \in \mathbf{O}_{big}} size_j \leq \frac{mH^*_{\max}}{\beta} \leq \frac{4m}{3\epsilon}$$

The first inequality is trivial since $size_j$ is always a strictly positive integer. Suppose now that the second inequality does not hold. Then the total volume of $\mathbf{O}_{big}$ is strictly more than $mH^*_{\max}$, that is strictly more than the total volume for all objects, which is not possible. The third inequality comes from the definition of β.

Combining the two equations, we obtain

$$K \leq 2^{\frac{4m^2}{3\epsilon}}$$

and an exhaustive search on $\mathbf{O}_{big}$ can be done in time $\mathcal{O}(2^{\frac{4m^2}{3\epsilon}})$.

The overall running time is then $\mathcal{O}(n\ln(n) + nm + 2^{\frac{4m^2}{3\epsilon}})$. □

A similar algorithm was proposed in Lemaire *et al.* (2003), based on the weaker Theorem 5. For this previous PTAS, every assignment of the "big" objects had to be considered, and the algorithm runs in time $\mathcal{O}(2^{\frac{2m(m-1)^2}{\epsilon}} nm)$. The principles were, however, the same.

The above PTAS, designed for multibin packing, has better performances on special cases, for instance partition or multiprocessor scheduling. However, it is outperformed by algorithms designed specifically for these particular cases. In particular, Hochbaum and Shmoys (1987) use an alternate approach based on dual approximation. They design an approximation scheme for multiprocessor scheduling that runs in time $\mathcal{O}((n/\epsilon)^{1/\epsilon^2})$ even if the number of processors m is not fixed. For further details, see Ausiello *et al.* (1999) and Hochbaum (1997).

Another important remark is that, using a different approach, a Fully Polynomial-Time Approximation Scheme (FPTAS) may be designed for problem $Bm|size_j|H_{\max}$. Indeed, the classical scaling technique (see Ausiello *et al.*, 1999 for details), used together with the pseudo-polynomial algorithm proposed in Lemaire *et al.* (2003), leads to a FPTAS that runs in time $\mathcal{O}(n(\frac{n^2}{\epsilon})^m)$.

However, the existence of our PTAS, based on a BF algorithm, has consequences on the asymptotic behaviour of this greedy algorithm.

For an instance I, let S_{DBF} be a solution computed by a DBF algorithm. For every ϵ such that $\mathbf{O}_{big}$ is empty, $H_{\max}(S_{DBF}) \leq (1+\epsilon)H^*_{\max}$. That is, for every ϵ such that

$$\begin{aligned} h_{\max} \leq \beta \quad &\Longleftrightarrow \quad h_{\max} \leq \epsilon \frac{3H_{approx}}{4} \\ &\Longleftrightarrow \quad \epsilon \geq \frac{h_{\max}}{H_{\max}(S_{DBF})} \end{aligned}$$

using $H_{approx} = H_{\max}(S_{DBF})$. Thus we have the following guarantee.

Property 10 *Let S be a solution computed by a DBF algorithm. Then,*

$$H_{\max}(S) \leq \left(1 + \frac{h_{\max}}{H_{\max}(S)}\right) H^*_{\max}$$

This property implies that DBF algorithms are *asymptotic* FPTAS if there is a bound on $h_{\max}$ (that is: for every ϵ there is a H such that $H^{BF}_{\max} \leq (1+\epsilon)H^*_{\max}$ for every instance such that $H^*_{\max} \geq H$). This has a practical consequence: BF algorithms are very good on large instances (i.e. with a lot of objects), especially if these objects are small. This is very similar to Corollary 1.

4. Conclusions

In this paper we consider a particular case of multibin packing, the $B|size_j|H_{\max}$ case. We first propose a greedy principle called the "best-fit rule" that leads to algorithms which are very fast and very efficient (both in theory and in practice). A refinement of this rule—sorting the objects by non-increasing height—leads to even better guarantees (and to improved experimental results), without increasing much the running time.

The approach is extended to design a polynomial time approximation scheme that solves the $Bm|size_j|H_{\max}$ case in polynomial time and within a factor of $(1+\epsilon)$, for any fixed number of bins m and any accuracy ϵ.

To go further with this work, two main issues would be the following.

- There is still a gap between Graham's or Coffman and Sethi's bounds for multiprocessor scheduling, and our bounds. It is an open matter if this gap can be filled or not.
- It would also be interesting to look for a polynomial time approximation scheme for an arbitrary m, maybe by adapting techniques in Hochbaum and Shmoys (1987). Notice that there cannot exist a fully polynomial approximation scheme for an arbitrary m, unless $\mathcal{P} = \mathcal{NP}$.

References

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A. and Protasi, M. (1999) *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, Berlin.

Błażewicz, J., Drozdowski, M., and Weglarz, J. (1994) Scheduling Multiprocessor Tasks—a survey. *International Journal of Microcomputer Applications*, **13**:89–97.

Chen, B., Potts, C. N. and Woeginger, G. J. (1998) A Review of Machine Scheduling: Complexity, Algorithms and Approximability. in: Du, D.-Z. and Pardalos, P. M. (Eds.), *Handbook of Combinatorial Optimization*, Kluwer, Dordrecht, pp. 21–169.

Chen, J. and Miranda, A. (2001) A Polynomial Time Approximation Scheme for General Multiprocessor Job Scheduling. *SIAM Journal on Computers*, **31**:1–17.

Coffman Jr, E. G. and Sethi, R. (1976) A generalized bound on LPT sequencing. *Revue Bleue d'AFCET, (R.A.O. Informatique)*, **10**:17–25.

Coffman Jr, E. G., Garey, M. R. and Johnson, D. S. (1997) Approximation Algorithms for Bin-Packing: a survey. in: Hochbaum, D. S. (Ed.), *Approximation Algorithms for NP-hard Problems*, PWS, Boston, MA, Chapter 2, pp. 46–93.

Drozdowski, M. (1996). Scheduling multiprocessor tasks—an overview. *European Journal of Operational Research*, **94**:215–230.

Graham, R. L. (1969) Bounds On Multiprocessor Timing Anomalies. *SIAM Journal of Applied Mathematics*, **17**:416–429.

Hochbaum, D. S. (Ed.) (1997) *Approximation Algorithms for NP-hard Problems*, PWS, Boston, MA.

Hochbaum, D. S. and Shmoys, D. B. (1987) Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results. *Journal of the Association for Computing Machinery*, **34**:144–162.

Kellerer, H., Pferschy, U. and Pisinger, D. (2004) *Knapsack Problems*, Springer, Berlin.

Khanna, S. (1997) A Polynomial Time Approximation Scheme for the SONET Ring Loading Problem *Bell Labs Technical Journal*, Spring, 36–41.

Lemaire, P., Finke, G. and Brauner, N. (2003) Packing of Multibin Objects. in: *IEPM'03, International Conference on Industrial Engineering and Production Management*, Vol. 1, pp. 422–431, Porto, Portugal. Updated version: `www-leibniz.imag.fr/~lemaire`.

Schrijver, A., Seymour, P. and Winkler, P. (1998) The Ring Loading Problem. *SIAM Journal on Discrete Mathematics*, **11**:1–14.

Educational Timetabling

CASE-BASED INITIALISATION OF METAHEURISTICS FOR EXAMINATION TIMETABLING

Sanja Petrovic, Yong Yang
School of Computer Science and Information Technology,
The University of Nottingham, Nottingham NG8 1BB, UK

Moshe Dror
University of Arizona, Tucson, AZ 85721, USA

Abstract Examination timetabling problems are traditionally solved by choosing a solution procedure from a plethora of heuristic algorithms based either on a direct construction principle or on some incremental improvement procedure. A number of hybrid approaches have also been examined in which a sequential heuristic and a metaheuristic are employed successively. As a rule, best results for a problem instance are obtained by implementing heuristics with domain-specific knowledge. However, solutions of this kind are not easily adoptable across different problem classes. In order to lessen the need for a problem-specific knowledge we developed a novel solution approach to examination timetabling by incorporating the case-based reasoning methodology. A solution to a given problem is constructed by implementing case-based reasoning to select a sequential heuristic, which produces a good initial solution for the Great Deluge metaheuristic. A series of computational experiments on benchmark problems were conducted which subsequently demonstrate that this approach gives comparable or better results than solutions generated not only by a single Great Deluge algorithm, but also the state-of-the-art approaches.

Keywords: case-based reasoning, heuristic selection, graph matching and retrieval.

1. INTRODUCTION

Examination timetabling problem is a difficult combinatorial optimisation problem. The task is to assign a set of examinations into a limited number of time periods and classrooms subject to constraints (Carter *et al.*, 1996). The constraints are usually divided into two categories: hard and soft constraints. Hard constraints are those that must not be violated. One such constraint is

that no student can attend two examinations at the same time. An examination timetable is considered to be feasible only when it meets all hard constraints. On the other hand, soft constraints are not essential to a timetable, but highly desirable. For example, students might wish to have a two or three day interval between two consecutive examinations. The quality of a timetable is measured in terms of its satisfaction of soft constraints. A good review of a variety of constraints that are usually imposed on examination timetabling is given in Burke *et al.* (1996).

A timetabling problem can be modelled by a graph where each vertex represents an examination while an edge represents a conflict between two examinations (e.g. two examinations have some students in common and therefore cannot be scheduled into the same period). Thus, a timetabling problem is analogous to a graph colouring problem when neglecting soft constraints and resource requirements, with colour-coding for time slots (Welsh *et al.*, 1967). The vertices of the graph have to be coloured in such a way so that no two adjacent vertices share the same colour. Note that finding a minimal number of colours to colour a graph is one of the classical NP-Complete problems (Garey and Johnson, 1977).

1.1 Examination Timetabling

Over the last 40 years, various approaches to examination timetabling have been developed. A number of review papers discuss approaches and research issues in examination timetabling (Burke and Petrovic, 2002; Carter, 1986; Carter and Laporte, 1996). Approaches based on applications of graph colouring heuristics for solving timetabling problems were widely employed in the early days of timetabling research (Carter, 1986; Foxley and Lockyer, 1968). The idea behind these heuristics is to schedule examinations sequentially commencing with the examinations estimated to be the most difficult to schedule and ending with the easiest ones. By the beginning of the 1990s, sequential heuristics had been superseded by various metaheuristics such as Tabu search and Simulated Annealing (SA), which take into consideration soft constraints and therefore produce more satisfactory solutions (Carter and Laporte, 1996).

Recently, there has been a growing interest in employing various sequential heuristics to generate initial solutions for metaheuristics. Saleh Elmohamed *et al.* (1998) used sequential heuristics which consider size of examinations to find a feasible solution and handled soft constraints by simulated annealing. Burke and Newell (1999) used sequential heuristics to decompose a large problem into several sub-problems, which were then solved by memetic algorithm. Di Gaspero and Schaerf (2001) designed sequential heuristics, which are hybridised with Tabu Search. The hybrid approaches (Burke and Newell 1999; Casey and Thompson, 2003; Merlot *et al.*, 2003; White *et al.*, 2004)

produced the best results on a number of benchmark problems, and represent the state of the art in timetabling. Sequential heuristics serve an important role in the successful subsequent implementation of metaheuristics because they cannot only shorten the search time but may also greatly enhance their performance (Burke *et al.*, 1998; Burke and Newell, 2002).

However, a successful development of such a metaheuristic is a difficult task since it usually involves incorporation of problem domain-specific knowledge. For example in a simulated annealing timetabling algorithm (Merlot *et al.*, 2003), a sophisticated neighbourhood structure (such as the Kempe chains), and an appropriate cooling schedule, which involves choosing a cooling formula and setting values for parameters such as initial temperature and cooling factor, have to be defined. Similarly, a Tabu Search timetabling algorithm (White *et al.*, 2004) requires an appropriate setting of parameters such as the length of the tabu list, the stopping criteria, and a candidate list strategy to restrict the neighbourhood size. Generally, the current approaches suffer from limitation in their applicability when faced with changes in problem description.

It is well known in the timetabling community that a solution procedure which generates good results at one university might perform poorly for timetabling problems in another university (Carter and Laporte, 1996). Naturally, the following question arises: which sequential heuristic should be used with a given metaheuristic for solving a timetabling problem at hand? In practice, a preferred solution for a given problem is usually obtained after appropriately selecting and "tailoring" both sequential heuristics and metaheuristics based on domain-specific knowledge of the problem.

In light of the above limitations, Burke *et al.* (2003a) applied a local search method, the Great Deluge algorithm (GDA), to solve timetabling problems. The "beauty" of the GDA is that it is much easier to develop a GDA algorithm compared to other metaheuristics, because it only requires one input parameter and therefore requires the least effort to "tailor" it for a given problem. It is worth noting that the authors showed that GDA is effective even by using a very simply defined neighbourhood structure. Burke and Newell (2002, 2003) extended this research further by applying an adaptive initialisation heuristic before running GDA. This adaptive heuristic firstly solves the problem a number of times in order to learn how to adjust the heuristic's parameters. Both methods produced best-published results on a range of benchmark problems.

It is desirable to develop a general timetabling system which works equally well for a variety of problem descriptions from different universities. Hyper-heuristic solution methodology, which is "an emerging methodology in search and optimisation" (Burke *et al.*, 2003a) aims at addressing these needs. Broadly speaking, the term of hyper-heuristics is defined as "the process of using (meta-)heuristics to choose (meta-)heuristics to solve the problem in hand"

(Burke *et al.*, 2003b). Terashima-Marín *et al.* (1999) presented a hyper-heuristic Evolutionary Approach for solving examination timetabling problems. The choices of different sequential heuristics, parameter value settings and the conditions for swapping sequential heuristics during the search process are encoded as chromosomes and evolved by a genetic algorithm. The timetable is built by the best chromosome founded by the genetic algorithm. Petrovic and Qu (2002) proposed a novel case-based hyper-heuristic to intelligently select sequential heuristics. A timetable is constructed by applying iteratively a number of sequential heuristics. The selection of a heuristic to improve the current partial solution is based on the performance of each heuristic in a similar situation. Their system requires a training process using the knowledge discovery techniques.

1.2 Case-Based Reasoning in Scheduling

Case-Based Reasoning (CBR) is an artificial intelligence methodology in which a new problem is solved by reusing knowledge and experience gained in solving previous problems (Leake, 1996; Kolodner, 1993). A case contains a description of the problem, and its solution. Cases are stored in a case base. The CBR process is divided into four phases (Aamodt and Plaza, 1994): retrieval of the case most similar to the new problem, reuse and revision of its solution, and inclusion of the new case in the case base.

Only a few applications of CBR to scheduling have been reported. The work on CBR so far can be classified into two categories. Approaches in the first category reuse the past problem solving methods or operators within a method for solving a new problem. Miyashita and Sycara (1995) built a CBR system CABINS, which improves sub-optimal solutions for job scheduling problems by applying iteratively a number of moves, chosen by CBR. A case in CABINS consists of a move operator and the context in which it proved to be useful. Schirmer (2000) applied CBR to choose the most suitable heuristic for solving different project scheduling problems. Petrovic *et al.* (2003a) developed a CBR system for nurse rostering problems, which stores scheduling repair knowledge of experts as cases and uses CBR to drive the constraint satisfaction procedure.

The second category of CBR approaches to scheduling reuse the whole solutions to a problem. Coello and Santos (1999) solved the real-time job scheduling problem by reusing solutions to similar problems. Similarly, Burke *et al.* (2001) established a CBR scheduler in which a new course timetabling problem is solved by revising the solution of a previously stored similar timetabling problem.

In this paper, (an early version of which appeared in Petrovic *et al.*, 2003b), we aim to develop a new approach which enhances the performance of GDA

on examination timetabling problems by intelligently applying an appropriate sequential heuristic for its initialisation. Section 2 briefly introduces a GDA and different sequential heuristics. The CBR approach developed for examination timetabling is described in Section 3. Section 4 presents experimental results and related discussion. Conclusions and future research work are given in Section 5.

2. GREAT DELUGE ALGORITHM AND SEQUENTIAL HEURISTICS

2.1 Great Deluge Algorithm

The GDA is a local search method introduced by Dueck (1993) that has been successfully applied to examination timetabling problems (Burke *et al.*, 2003b). It represents a modification of the SA approach (Kirkpatrick *et al.*, 1983). Apart from accepting a move that improves the solution quality, GDA also accepts a move that results in a decrease of the solution quality as long as the decrease of the solution quality is smaller than a given upper boundary value, referred to as "water-level". In this work, the water-level is initially set to be the objective function value of the initial solution multiplied by a predefined factor. The neighbouring solutions of the current solution are obtained by moving an examination to a different time slot. After each move, the water-level is iteratively decreased by a fixed rate, which is equal to the initial value of the water-level divided by the time that is allocated to the search (expressed as the total number of moves). Not surprisingly, the GDA produces better solutions with the prolongation of the search time of the algorithm. This does not hold for a number of other local search algorithms where the user does not control the search time.

2.2 Sequential Heuristics

A variety of sequential heuristics can be used to construct initial solutions for GDA. They sort examinations based on the estimated difficulty of their scheduling. A number of sequential heuristics are briefly described as follows.

1 Largest degree (LD). Examinations with the largest number of conflicts are scheduled first.

2 Largest enrolment (LE). A modification of LD: it schedules examinations with the largest student enrolment first.

3 Largest colour degree (CD). A dynamic version of LD: it prioritises examinations by the largest number of conflicts with other examinations, which have already been scheduled.

4 Largest weighted degree (LWD). LWD is a combination of LD and LE. The highest priority is given to the examination with the largest sum of the weighted conflicts, where each conflict is weighted by the number of students who are enrolled in both examinations.

5 Least saturation degree (SD). Examinations with the least number of available periods for placement should be scheduled first (Brelaz, 1979).

These sequential heuristics can be enriched in a variety of ways. The most common ones are listed below:

1 Maximum clique detection (MCD). The maximum clique is the largest completely connected subgraph of a graph. The cardinality of the maximum clique determines the lower bound on the number of time periods needed for the timetable (Carter, 1986). Finding the maximum clique is an NP-Complete problem (Garey and Johnson, 1977). Vertices of the maximum clique are regarded as the most difficult examinations to schedule and therefore should be scheduled first (Carter *et al.*, 1996). In this research, a tabu search heuristic approach proposed by Gendreau *et al.* (1993) was implemented to find the vertices in the maximum clique of a given graph.

2 Adding random elements (ARE). The examination to be scheduled next is selected from a subset of randomly chosen examinations (Burke *et al.*, 1998). The size of the subset is given as the percentage of the full set of examinations.

3 Backtracking (BT). Some examinations cannot be assigned to any time period without violating hard constraints. In order to schedule these examinations, some previously scheduled examinations that are in conflict with the examinations at hand are rescheduled. Several rules are used to prevent cycles (Laporte and Desroches, 1984).

Sequential heuristics investigated in this research are hybridized with MCD, and/or BT, and/or ARE where 30%, 60% or 90% of examinations not yet scheduled are chosen randomly to form the subset of examinations to choose from. Selecting a suitable heuristic to generate an initial solution for the GDA is of high importance, because it can significantly affect the quality of the final solution.

3. CBR SYSTEM FOR EXAMINATION TIMETABLING

It is not an easy task to select an appropriate sequential heuristic to construct a good initial solution for GDA. It would be computationally very expensive to

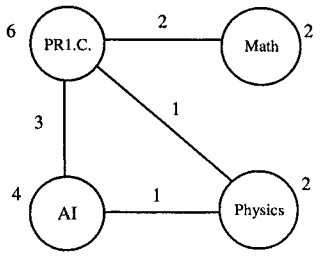
try every combination of sequential heuristics and GDA. Thus, we developed a CBR system, which selects a sequential initialisation heuristic for GDA in order to produce a high quality solution for a given problem. The rationale behind this study is that given an effective hybridisation of a certain sequential heuristic and GDA for a specific timetabling problem, it is likely that it will also be effective for a "similar" problem.

In our CBR system, a case memorises an examination timetabling problem and an effective sequential heuristic, which has generated an appropriate initial solution for GDA. For solving a new input timetabling problem, the sequential heuristic of the most similar case is proposed. The main research issue is how to define the "similarity" measure between two timetabling problems.

3.1 Case Representation

In this section, we explain how the important features of examination timetabling problems are incorporated into the case representation. An examination timetabling problem is represented by an undirected weighted graph $G = (V, E, \alpha, \beta)$, where V is the set of vertices that represent examinations, $E \subseteq V \times V$ is the finite set of edges that represent conflicts between examinations, $\alpha : V \mapsto N^+$ assigns positive integer weights to vertices that correspond to the number of students enrolled in the examination, and $\beta : E \mapsto N^+$ is an assignment of weights to edges which correspond to the number of students enrolled in two examinations that are in conflict. $|V|$ is used to denote the cardinality of the set V For illustration purpose, a simple example is given in Figure 1. In this figure the weight of Math is 2 because two students are enrolled in this course. The edge connecting AI and Physics is assigned weight 1 because there is one student who is enrolled in both examinations. Important features of the timetabling problem, such as number of examinations, number of enrolments, and number of constraints, are incorporated into the weighted graph case representation. Moreover, the weighted graph case representation is capable of describing highly inter-connected constraints that are imposed between examinations and on examinations themselves.

A solution to an examination timetabling problem is denoted by a vector $S = (s_1, s_2, \ldots, s_{|V|})$, where $s_n, n = 1, \ldots, |V|$, represents the time period assigned to the examination n. A feasible (conflict free) solution is a solution in which for any two vertices $a \in V$ and $b \in V$, then s_a, must be different from s_b if $(a, b) \in E$. The cost function often used in timetabling community for solution evaluation soft constraints was proposed by Carter *et al.* (1994). The common cost function enables comparison of quality of solutions produced by different approaches. The cost function gives a cost w_s to a solution whenever a student has to sit two examinations s periods apart. Costs that are used are $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$, $w_5 = 1$. The cost function sums all the



Student ID.	Enrolled Examination
2000101	AI; PR1 C.
2000102	AI; PR1 C.
2000103	AI; PR1 C.
2000104	AI; Physics.
2000105	Math; PR1 C.
2000106	Math; PR1 C.
2000107	Physics; PR1 C.

Figure 1. An examination timetabling problem represented by a graph.

costs of each student and divides the obtained sum by the number of students. The value obtained is the average cost for each student.

A case C can be formally represented by an ordered pair (G, H), where G is the graph representation of an examination timetabling problem, while H is the sequential heuristic that produced an initial solution appropriate for GDA.

3.2 Similarity Measure

An adequate definition of similarity measure is of great importance for a CBR system because it enables the retrieval of the case that is most closely related to the new problem. Since weighted graphs are used to represent timetabling problems, the retrieval of the most similar case from the case base requires solving a graph isomorphism problem, which is known to be NP-Complete (Garey and Johnson, 1977).

The following notation will be used. We denote a new timetabling problem to be solved (a query case) by C_q and a source case in the case base by C_s, while their weighted graphs are denoted by $G_q = (V_q, E_q, \alpha_q, \beta_q)$ and $G_s = (V_s, E_s, \alpha_s, \beta_s)$, respectively. In order to compute the similarity degree between C_q and C_s, a vertex-to-vertex correspondence has to be established that associates vertices in V_q with those in V_s. The correspondence is represented by the function $f : V_q \rightarrow V_s$.

Latin and Greek letters are used to denote vertices and edges in G_q and G_s, respectively. For instance, $f(a) = \chi$ denotes that vertex $a \in V_q$ is mapped to the vertex $\chi \in V_s$ by the correspondence f. In this research, the computation of the similarity degree between pairs of vertices, edges and graphs proposed by Wang and Ishii (1997) is modified to include the concept of weights employed in our problem.

The similarity degree between two vertices in G_q and G_s determined by the correspondence f is denoted by $DS_f(a, \chi)$:

$$DS_f(a,\chi) = \begin{cases} \text{Min}(\alpha_q(a), \alpha_s(\chi)), & \text{if } f(a) = \chi \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Similarly, $DS_f(x, \varphi)$ represents the similarity degree between two edges determined by the correspondence f, where $x = (a, b) \in E_q$ and $\gamma = (\chi, \delta) \in E_s$:

$$DS_f(x,\gamma) = \begin{cases} \text{Min}(\beta_q(x), \beta_s(\gamma)), & \text{if } f(a) = \chi \text{ and } f(b) = \delta \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

We use the label ϕ to denote an extraneous vertex or edge in a graph, which is not mapped by the correspondence f. We set $DS_f(a, \phi)$, $DS_f(\phi, \chi)$, $DS_f((a, b), \phi)$ and $DS_f(\phi, (\chi, \delta))$ to be equal to 0. Finally, the similarity degree $DS_f(G_q, G_s)$ between G_q and G_s determined by the correspondence f is calculated in the following way:

$$DS_f = (G_q, G_s) = \frac{F_v + F_e}{M_v + M_e} \tag{3}$$

where

$$F_v = \sum_{a \in Vq} \sum_{\chi \in V_s} DS_f(a, \chi) \tag{4}$$

$$F_e = \sum_{x \in Eq} \sum_{\gamma \in E_s} DS_f(x, \gamma) \tag{5}$$

$$M_v = \text{Max}\left(\sum_{a \in Vq} \alpha_q(a), \sum_{\chi \in V_s} \alpha_s(\chi) \right) \tag{6}$$

$$M_e = \text{Max}\left(\sum_{x \in Eq} \beta_q(x), \sum_{\gamma \in E_s} \beta_s(\gamma) \right) \tag{7}$$

Note that the value of $DS_f(G_q, G_s) \in [0, 1]$ is subject to correspondence f. The task is to find the correspondence f that yields as high a value of $DS_f(G_q, G_s)$ as possible.

3.3 Case Retrieval

The goal of the case retrieval is to find a case in the case base whose graph is the most structurally similar to that of the new problem. The retrieval of the

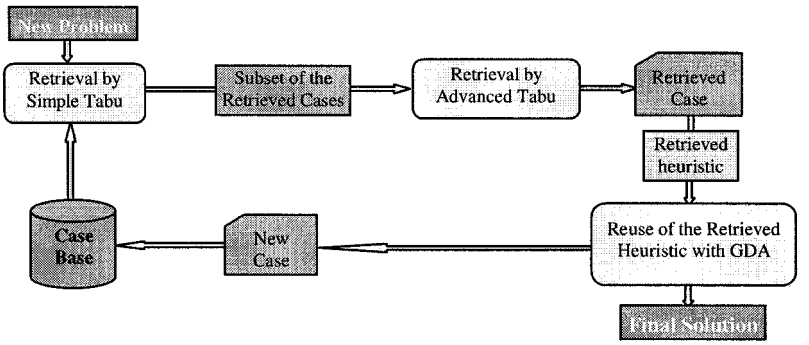


Figure 2. Architecture of the CBR system for heuristic initialisation of meta-heuristics.

graph-structured cases is a difficult process. Firstly, it is difficult to develop a valid indexing scheme to manage the organisation of graph-structured cases in the case base. Secondly, there is an expensive computational cost for calculating the similarity degree between two cases that involves graph matching.

For case retrieval we employ a two-stage Tabu Search described in more detail in Petrovic *et al.* (2002). The search procedure is guided by the short-term and long-term adaptive memories (Glover and Laguna, 1997). The short-term memory is used to prevent the search process from cycling by forbidding moves, which have been made recently. The long-term memory holds the history of all moves and is used to guide the search process to vicinities of elite solutions or regions that have not yet been explored. In order to reduce the computational cost required in the retrieval process, it is divided into two phases. Firstly, the simple Tabu Search with its short-term memory is used to quickly select a subset of cases from the case base considered to be similar enough to the new problem. Then the advanced Tabu Search enriched with long-term memory is used for the final more precise retrieval of the case.

3.4 Architecture of the CBR System

The architecture of our CBR system is depicted in Figure 2. The retrieval process is performed by the simple and advanced Tabu Search algorithms. The sequential heuristic, which has been shown to be the most appropriate for generating the initial solution for GDA for solving the retrieved case, is then proposed for the initialisation of GDA to be applied to the new problem. Once the problem is solved, the new problem together with the retrieved sequential heuristic will be stored as a new case in the case base.

4. EXPERIMENTS

The purpose of the designed experiments is twofold: evaluation of effectiveness and efficiency of the case retrieval and evaluation of system performance on a range of real world examination timetabling problems. Experiments were run on a PC with an Athlon 1400 Mhz CPU and 256 MB RAM.

4.1 Description of Seeding Cases

A number of real-world examination problems that are often used as benchmark problems within the timetabling community are used for the construction of cases, which will form a case base. The characteristics of these timetabling problems are given in Table 1. The conflict matrix is used to represent conflicts between pairs of examinations. Rows and columns of the matrix represent examinations, while each element of the matrix shows the number of students common for a pair of examinations. The density of the conflict matrix is calculated as the ratio of the number of exams in conflict to the total number of exams.

Seven benchmark problems (ear-f-83, hec-s-92, kfu-s-93, lse-f-91, sta-f-83, tre-s-92, and yor-f-83) and their specially designed variations were used to seed the case base. Variation problems were created by adding or deleting a given a number of students and examinations from the benchmark problem. Each variation problem is denoted by x/y, where x gives the percentage of both examinations and students of a benchmark problem which were added, and y gives the same percentage of those which were deleted. We defined five categories of variation problems: $5/15$, $5/10$, $5/5$, $10/5$, $15/5$. For example, the expression $5/15$ denotes a variation problem that was created by randomly adding and deleting 5% and 15% of the numbers of examinations and students of an associated corresponding benchmark problem, respectively. Two case bases of different sizes were created. The large case base contains ten variations of each seeding benchmark problem (two for each category). Thus it contains 77 cases (seven benchmark problems and 70 variations). The small case base contains five variations of each benchmark problem, which gives in total 42 cases (seven benchmark problems and 35 variations).

For each seeding problem, all possible sequential heuristics were used to produce initial solutions for GDA. The sequential heuristic, which led to the best final solution, was stored in the case. In this experiment, GDA was run for 20,000,000 iterations while water-level was set to be 1.3. The number of iterations was set empirically and the "water-level" was taken from (Burke *et al.*, 2003b).

The modifications of benchmark problems were used as seeding cases, because we noticed that heuristics proven to be the best for GDA initialisation for variation problems could be different from that of the original benchmark

Table 1. The benchmark problems used for seeding the case base.

Data	Institution	Periods	Number of exams	Number of students	Number of enrolments	Density of conflict matrix
Car-f-92	Carleton University, Ottawa	32	543	18,419	55,522	0.14
Car-s-91	Carleton University, Ottawa	35	682	16,925	56,877	0.13
Ear-f-83	Earl Haig Collegiate Institute, Toronto	24	190	1,125	8,109	0.29
Hec-s-92	Ecole des Hautes Etudes Commercials, Montreal	18	81	2,823	10,632	0.20
Kfu-s-93	King Fahd University Dharan	20	461	5,349	25,113	0.06
Lse-f-91	London School of Economics	18	381	2,726	10,918	0.06
Rye-s-93	Ryeson University, Toronto	23	486	11,483	45,051	0.07
Sta-f-83	St Andrew's Junior High School, Toronto	13	139	611	5,751	0.14
Tre-s-92	Trent University, Peterborough, Ontario	23	261	4,360	14,901	0.18
Ute-s-92	Faculty of Engineering, University of Toronto	10	184	2,750	11,793	0.08
Uta-s-92	Faculty of Arts and Science, University of Toronto	35	622	21,276	58,979	0.13
Yor-f-83	York Mills Collegiate Institute, Toronto	21	181	941	6,034	0.27

problem. In addition, the system performance will be improved by adding timetabling problems with different graph structures.

The aim of the first experiment is to show how often each heuristic appears to be the best for GDA in the timetabling problems of the large case base (77 cases). Note that the best initial solution (with respect to the cost function) does not necessarily lead to the best final solution produced by the GDA.

Figure 3 shows how many times each of the sequential heuristics appears in 77 seeding cases. A triplet (a, b, c) is assigned to each heuristic SD, LD, CD, LE and LWD, which denotes whether the heuristic was enriched with maximum clique detection ($a = 1$, otherwise $a = 0$), backtracking ($b = 1$, otherwise $b = 0$), and/or adding random element ($c = 30\%$, or 60%, or 90%). We can notice that although some heuristics are used more than the others, there is no single heuristic that outperforms all the rest.

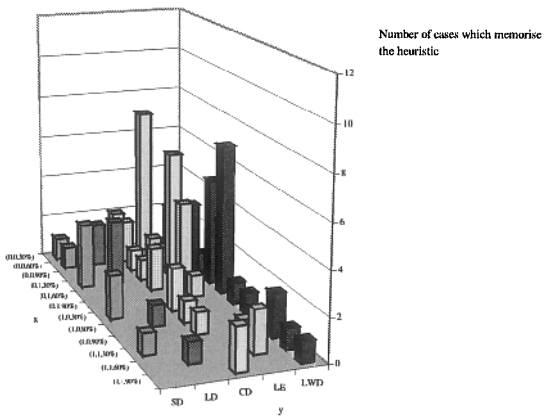


Figure 3. Use of heuristics in GDA initialisation.

4.2 Evaluation of the Case Retrieval Process

Since the retrieval process uses the simple and advanced Tabu Search sequentially, our second set of experiments investigates how precisely and with what computational cost for each of the approaches the similarity degree between two timetabling problems can be calculated. Due to the NP-Complete nature of the graph isomorphism problem, it is not possible to evaluate the performance of the two Tabu Search algorithms on two randomly chosen graphs. In this study we used the following method that is based on the experiments of Luo and Hancock (2001).

In our experiments, for each graph G of the seeding case, a copy of it is denoted by G_c. The similarity degree between G and G_c is 1 when each vertex in G is mapped to its corresponding vertex in G_c. This vertex-to-vertex mapping is considered as the ideal one. Then for each pair of G and G_c, both simple and advanced Tabu Search algorithms were run where the initial solution was a random vertex-to-vertex mapping. We evaluate the performance of our Tabu Search algorithms by examining whether they could find this ideal mapping between G and G_c, or how closely they approach it. We use DS_{tabu} to denote the similarity degree obtained after running a Tabu search algorithm. The closer the value of DS_{tabu} to 1, the better the vertex-to-vertex mapping that has been found.

Figure 4 shows the performance and search time of the two Tabu Search algorithms, respectively. A test running is terminated either when a solution has not improved for 2,000 moves or when an ideal mapping has been found. Circles and squares show the average value of DS_{tabu} obtained for the bench-

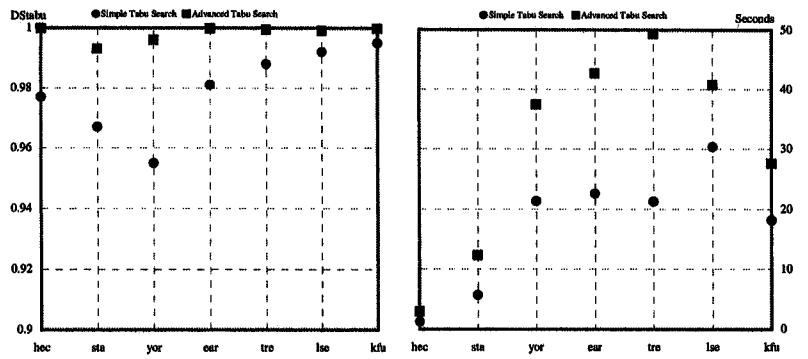


Figure 4. Performance of the simple and advanced Tabu Search.

mark problem (denoted on the x-axis) and its 10 variation problems by simple and advanced Tabu Search algorithm, respectively. Benchmark problems are sorted by their number of examinations in ascending order.

In Figure 4, it can be seen that the simple Tabu Search can obtain DS_{tabu} in the range from 0.95 to 0.995. The advanced Tabu Search could find the "ideal" mapping of vertices in five out of seven benchmark problems. For the remaining two problems Sta-f-83 and Yor-f-83, the advanced Tabu Search did not provide the "ideal" mapping, but the average values of DS_{tabu} are still high. On the other hand, the time required for advanced Tabu Search is not significantly higher than for simple Tabu Search. This justifies the implementation of the advanced Tabu Search within the retrieval process.

The experimental results show that the simple Tabu Search is capable of calculating the similarity degree approximately within a relatively short time and is used to filter the cases from the case base and to pass a predefined number of cases with the highest similarity degree with the new problem to the advanced tabu search. The advanced Tabu Search then spends more time finding mapping for graph isomorphism on the smaller set of cases.

4.3 Performance on Benchmark Problems

The purpose of the third set of experiments is to test performance of our CBR system on benchmark problems. In addition we want to investigate whether the size of the case base has an impact on the performance of the system. For each of the five benchmark problems (Car-f-92, Car-s-91, Rye-s-93, Uta-s-92, and Ute-s-92), the problem was solved twice by using the small and the large case base. For each of the other seven seeding bench-

Table 2. Results obtained using case bases of different sizes.

Data	Small case base				Large case base			
	Time (s)		Cost		Time (s)		Cost	
	Retrieval	Run GDA	Best	Avg.	Retrieval	Run GDA	Best	Avg.
Car-f-92	2274	1231	3.97	4.08	3772	1231	3.97	4.08
Car-s-91	3301	1321	4.54	4.65	5788	1321	4.54	4.65
Ear-f-83	1164	1250	34.49	36.06	2158	811	34.49	36.06
Hec-s-92	377	1540	10.92	11.29	699	1540	10.92	11.29
Kfu-s-93	1982	735	14.82	15.11	3701	735	14.82	15.11
Lse-f-91	1769	587	11.48	**11.60**	3077	586	10.60	**10.83**
Rye-s-93	1775	872	9.0	**9.66**	3097	930	9.0	**9.39**
Sta-f-83	373	695	160.29	**160.83**	673	680	159.89	**160.49**
Tre-s-92	1878	715	7.96	8.09	2936	786	7.96	8.09
Ute-s-92	430	508	25.74	**26.22**	786	637	25.64	**26.09**
Uta-s-92	2534	1271	3.26	3.29	4768	1271	3.26	3.29
Yor-f-83	1096	1300	36.82	**37.26**	2006	1320	36.69	**37.08**

mark problems, it was also used as a new problem and solved twice by the two case bases (the problem itself and its variation problems were removed from the case base in the retrieval process). Therefore, for each seeding benchmark problem, the small case base includes the other six benchmark problems and 30 associated variation problems; the large case base includes the other six benchmark problems and 60 associated variation problems. We ran each experiment five times to obtain the average results. The results are summarised in Table 2.

We can see that, as expected, the retrieval of more similar cases can lead to better solutions. For five problem instances Lse-f-91, Rye-s-93, Sta-f-83, Ue-s-92 and Yor-f-83 (highlighted by bold characters), the large case base yielded better solutions, while for the remaining instances both case bases gave the same solutions. The price to be paid is of course longer time spent on the case retrieval, which is proportional to the number of cases.

Table 3 provides the comparison of the average results obtained by three other GDA initialisation approaches previously tried in the timetabling literature on these benchmark problems, namely GDA where SD is used to provide the initial solution (Burke *et al.*, 2003b), GDA where initial solutions drive the adaptation of the parameters of the algorithm throughout the search (Burke and Newell, 2003; Burke and Newell, 2002), and GDA where the combination of SD, MCD and BT is used to construct an initial solution (Carter *et al.*, 1996). Again GDA was run for 200×10^6 iterations, because that was the number of iterations used in the methods that we compare our approach with. For illustration purpose, we also provide the time spent on the search (in seconds).

Table 3. Comparison of results for benchmark problems obtained by different initialisation of GDA.

Data	SD			Adaptive			SD & MCD & BT			CBR			
	GDA Time	Best Cost	Avg. Cost	GDA Time	Best Cost	Avg. Cost	GDA Time	Best Cost	Avg. Cost	Retrieval Time	GDA Time	Best Cost	Avg. Cost
Car-f-92	1120	4.03	4.07	416	–	4.10	1220	3.97	**4.04**	3772	1231	3.97	4.08
Car-s-91	1400	4.57	**4.62**	681	–	4.65	1441	4.62	4.66	5788	1321	4.54	4.65
Ear-f-83	806	34.85	**36.04**	377	–	37.05	767	33.82	36.26	2158	811	34.49	36.06
Hec-s-92	1471	11.27	12.43	516	–	11.54	1411	11.08	11.48	699	1540	10.92	**11.29**
Kfu-s-93	843	14.33	14.64	449	–	**13.90**	996	14.35	14.62	3701	735	14.82	15.11
Lse-f-91	646	11.61	11.65	341	–	**10.82**	675	11.57	11.94	3077	586	10.60	10.83
Rye-s-93	845	9.19	9.66	–	–	–	881	9.32	9.50	3097	930	9.0	9.39
Sta-f-83	675	165.12	169.7	418	–	168.73	674	166.07	166.31	673	680	159.89	**160.49**
Tre-s-92	907	8.13	8.29	304	–	8.35	751	8.19	8.27	2936	786	7.96	**8.09**
Ute-s-92	716	25.88	26.05	324	–	**25.83**	653	25.53	26.02	786	637	25.64	26.09
Uta-s-92	1070	3.25	3.30	517	–	**3.20**	1101	3.24	3.31	4768	1271	3.26	3.29
Yor-f-83	1381	36.17	**36.59**	695	–	37.28	1261	36.31	37.27	2006	1320	36.69	37.08
Rank			2.58			2.55			2.5				**2.08**

Although each algorithm was allocated the same number of iterations, the time is different due to computers of different characteristics. The table shows the average time spent on the search (each problem instance was solved five times).

In order to examine the performance of each initialisation method further, we also show the rank of the average cost that a method obtained on problem instances. This evaluation method was introduced by White *et al.* (2004). For example, the rank on the problem instance Car-f-92 is computed as: SD, 2; Adaptive, 4; MCD&BT&SD, 1; CBR, 3. The bottom row of Table 3 shows the average of the ranks for the 12 problems (excluding the rank of Rye-s-93 for the Adaptive initialisation method) of four different approaches.

We can see that our CBR system outperforms other initialisation methods for GDA. The CBR initialisation obtained the best rank (2.08) among all the methods. More significantly, we obtained the best average results for four benchmark problems (highlighted by bold characters). For the remaining seven problems, the other three methods only slightly outperformed our approach except for the problem instance Kfu-s-93. It is evident that our CBR system spent additional time on the case retrieval. However, the quality of the obtained results justifies the time spent on the selection of an appropriate heuristic, which determines a good starting point for the GDA.

Finally, we compare our results with those produced by the state-of- the-art timetabling metaheuristics: SA (Merlot *et al.*, 2003), Tabu search (White *et al.*, 2004), and GRASP (Casey and Thompson, 2003) by Table 4.

Table 4. Results for benchmark problems obtained by different timetabling approaches.

Data	SA			Tabu			GRASP			CBR			
	Time	Best Cost	Avg. Cost	Time	Best Cost	Avg. Cost	Time	Best Cost	Avg. Cost	Retrieval Time	GDA Time	Best Cost	Avg. Cost
Car-f-92	233	4.3	4.4	–	4.63	4.69	–	4.4	4.7	3772	1231	**3.97**	**4.08**
Car-s-91	296	5.1	5.2	–	5.73	5.82	–	5.4	5.6	5788	1321	**4.54**	**4.65**
Ear-f-83	26	35.1	35.4	–	45.8	45.6	–	34.8	**35.0**	2158	811	**34.49**	36.06
Hec-s-92	5.4	**10.6**	**10.7**	–	12.9	13.4	–	10.8	10.9	699	1540	10.92	11.29
Kfu-s-93	40	**13.5**	**14.0**	–	17.1	17.8	–	14.1	14.3	3701	735	14.82	15.11
Lse-f-91	35	**10.5**	11.0	–	14.7	14.8	–	14.7	15.0	3077	586	10.6	**10.83**
Rye-s-93	70	**8.4**	**8.7**	–	11.6	11.7	–	–	–	3097	930	9.0	9.39
Sta-f-83	5	157.3	157.4	–	158	158	–	**134.9**	**135.1**	673	680	159.89	160.49
Tre-s-92	39	8.4	8.6	–	8.94	9.16	–	8.7	8.8	2936	786	**7.96**	**8.09**
Ute-s-92	9	**25.1**	**25.2**	–	29.0	29.1	–	25.4	25.5	786	637	25.64	26.09
Uta-s-92	233	3.5	3.6	–	4.44	4.49	–	–	–	4768	1271	3.26	3.29
Yor-f-83	30	37.4	37.9	–	42.3	42.5	–	37.5	38.1	2006	1320	**36.69**	**37.08**

Table 5. Average of the ranks for benchmark problems obtained by different approaches.

Approaches	SA	Tabu	GRASP	SD GDA	Adaptive GDA	SD & MCD & BT GDA	CBR GDA
Average Rank	**3.00**	6.17	4.0	3.58	3.36	3.67	3.08

We obtained six best average costs and six best costs (highlighted) out of 12 benchmark problems.

Table 5 shows the average of the ranks for the 12 problem instances. Due to the incomplete results in Burke and Newell (2002) and Casey and Thompson (2003), we exclude the rank of Rye-s-93 for the Adaptive GDA method, and the rank of Rye-s-93 and Uta-f-92 for the GRASP method.

We can see that SA and the GDA initialised by CBR obtained the best rank (3.00) and the second best rank (3.08) among the seven different approaches investigated. However, opposite to SA our approach does not require parameter "tuning" for a particular timetabling problem and design of appropriate neighbourhood structure. In addition, the experience gained in solving one timetabling problem is not wasted but can be used in solving new similar timetabling problems.

5. CONCLUSIONS

In this paper we have presented a case-based reasoning system, which selects an appropriate sequential heuristic for generating an initial solution for

Great Deluge algorithm (GDA). We have shown that with an appropriate definition of "similarity" measure, such initialisation of GDA provides high-quality solutions for a range of real-world problems. One of the insights of this study is that our CBR system significantly contributes to the attempt of building a general metaheuristic framework for timetabling. Usually in metaheuristics, a random initialisation is employed or a thorough investigation of heuristics needs to be performed, which is useful only for a given problem instance. In this paper, we demonstrated that knowledge gained in initialisation of one timetabling problem can be used for solving new similar timetabling problems.

The developed CBR system examined in this paper contains cases with complex structures represented by weighted graphs. We have shown that the two-phase Tabu Search approach is capable of retrieving graph-structured cases where graphs are of large size and the case base contains hundreds of cases. We believe that the graph-structured case representation, the similarity measure, and the proposed case retrieval are applicable to other domains such as job shop scheduling, planning and other CBR applications.

The results obtained so far provide us with a good foundation for the development of a more general CBR system for solving timetabling problems. Our future research direction will include improvements aimed to shorten the required time for the case retrieval. We will also investigate hierarchical case representation that would enable the case retrieval process to examine only a subset of the case base. Finally, we will investigate the hybridisation of sequential heuristics and other local search methods such as tabu search and simulated annealing.

Acknowledgments

The authors wish to thank Dr Jim Newall for offering the source code of the timetabling library, and the anonymous reviewers for their valuable remarks on this work.

References

Aamodt, A. and Plaza, P. (1994) Case-based reasoning: foundational issues, methodological variations and system approaches. *The European Journal on Artificial Intelligence*, **7**:39–59.

Brelaz, D. (1979) New methods to color the vertices of a graph. *Communication of ACM*, **22**:251–256.

Burke, E. K. and Newell, J. P. (1999) A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, **31**:63–74.

Burke, E. K. and Newall, J. P. (2002) Enhancing Timetable Solutions with Local Search Methods. In *The Practice and Theory of Automated Timetabling IV*, Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 195–206.

Burke, E. K. and Newell, J. P. (2003) Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of Operations Research*, accepted for publication.

Burke, E. K. and Petrovic, S. (2002) Recent research directions in automated timetabling. *European Journal of Operational Research*, **140**:266–280.

Burke, E. K., Elliman, D. G., Ford, P. H. and Weare, R. F. (1996) Examination timetabling in british universities—A survey. In *The Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 76–92.

Burke, E. K., Newall, J. P. and Weare, R. F. (1998) Initialisation strategies and diversity in evolutionary timetabling. *Evolutionary Computation Journal*, **6**:81–103.

Burke, E. K., Newell, J. P. and Weare, R. F. (1998) A simple heuristically guided search for the timetable problem. In *Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems*, University of La Laguna, pp. 574–579.

Burke, E. K., MacCarthy, B., Petrovic, S. and Qu, R. (2001) Case-based reasoning in course timetabling: an attribute graph approach. In *Proceedings of 4th International Conference on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, Vol. 2080, Springer, Berlin, pp. 90–104.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S. (2003a) Hyper-heuristics: an emerging direction in modern search technology. In *Handbook of Meta-Heuristics*, Chapter 16, pp. 457–474, Kluwer, Dordrecht.

Burke, E. K., Bykov, Y., Newall, J. P. and Petrovic, S. (2003b) A time-predefined local search approach to exam timetabling problems. *IIE Transactions on Operations Engineering*, **36**:509–528.

Carter, M. W. (1986) A survey of practical applications on examination timetabling. *Operations Research*, **34**:193–202.

Carter, M. W. and Laporte, G. (1996) Recent developments in practical examination timetabling. In *The Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 3–21.

Carter, M. W., Laporte, G. and Chinneck, J. W. (1994) A general examination scheduling system. *Interfaces*, **24**:109–120.

Carter, M. W., Laporte, G. and Lee, S. Y. (1996) Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society*, **47**:373–383.

Casey, S. and Thompson, J. (2003) GRASPing the examination scheduling problem. In *The Practice and Theory of Automated Timetabling IV*, Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 232–246.

Coello, J. M. A. and Santos, R. C. (1999) Integrating CBR and heuristic search for learning and reusing solutions in real-time task scheduling. In *Proceedings of 3rd International Conference on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, Vol. 1650, Springer, Berlin, pp. 89–103.

Di Gaspero, L. and Schaerf, A. (2001) Tabu search techniques for examination timetabling. In *Proceedings of Practice and Theory of Automated Timetabling III*, Lecture Notes in Computer Science, Vol. 2079, Springer, Berlin, pp. 104–117.

Dueck, G. (1993) New optimization heuristics. *Journal of Computational Physics*, **104**:86–92.

Foxley, E. and Lockyer, K. (1968) The construction of examination timetable by computer. *The Computer Journal*, **11**:264–268.

Garey, M. R. and Johnson, D. S. (1977) Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco.

Gendreau, M., Soriano, P. and Salvail, L. (1993) Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, **41**:385–403.

Glover, F. and Laguna, M. (1997) Tabu Search. Kluwer, Dordrecht.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983) Optimisation by simulated annealing. *Science*, **220**:671–680.

Laporte, G. and Desroches, S. (1984) Examination timetabling by computer. *Computers and Operations Research*, **11**:351–360.

Leake, D. B. (1996) CBR in context: the present and future. In *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, D. Leake (Ed.), AAAI Press/MIT Press, Menlo Park, CA.

Luo, B. and Hancock, E. R. (2001) Structural graph matching using the em algorithm and singular value decomposition. *IEEE Transactions Analysis and Machine Intelligence*, **23**:1120–1136.

Kolodner, J. (1993) Case-Based Reasoning. Morgan Kaufmann, San Mateo, CA.

Merlot, L. T. G., Boland, N., Hughs, B. and Stucky, P. J. (2003) A hybrid algorithm for the examination timetabling problem. In *The Practice and Theory of Automated Timetabling IV*, Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 207–231.

Miyashita, K. and Sycara, K. (1995) CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence*, **76**:377–426.

Petrovic, S. and Qu, R. (2002) Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems. In *Proceedings of Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies*, Frontiers in Artificial Intelligence and Applications, Vol. 82, IOS Press, Amsterdam, pp. 336–340.

Petrovic, S., Kendall, G. and Yang, Y. (2002) A tabu search approach for graph-structured case retrieval. In *Proceedings of the STarting Artificial Intelligence Researchers Symposium*, IOS Press, Amsterdam, pp. 55–64.

Petrovic, S., Beddoe, G. R. and Berghe, G. V. (2003a) Storing and adapting repair experiences in employee rostering. In *Practice and Theory of Automated Timetabling IV*, Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 149–166.

Petrovic, S., Yang, Y. and Dror, M. (2003b) Case-based initialisation of metaheuristics for examination timetabling. In *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, pp. 137–155.

Saleh Elmohamed, M. A., Coddington, P. and Fox, G. (1998) A comparison of annealing techniques for academic course scheduling. In *The Practice and Theory of Automated Timetabling II*, Lecture Notes in Computer Science, Vol. 1408, Springer, Berlin, pp. 92–112.

Schirmer, A. 2000 Case-based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics*, **47**:201–222.

Terashima-Marín, H., Ross, P. and Valenzuela-Rendón, M. (1999) Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the Genetic and Evolutionary Conference*, pp. 635–642.

Wang, Y. and Ishii, N. (1997) A method of similarity metrics for structured representations. *Expert Systems with Applications*, **12**:89–100.

Welsh, D. J. A. and Powell, M. B. (1967) An upper bound on the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, **10**:85–86.

White, G. M. Xie, B. S. and Zonjic, S. (2004) Using tabu search with longer-term memory and relaxation to create examination timetables. *European Journal of Operational Research*, **153**:80–91.

AN INVESTIGATION OF A TABU-SEARCH-BASED HYPER-HEURISTIC FOR EXAMINATION TIMETABLING

Graham Kendall and Naimah Mohd Hussin
Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
School of Computer Science and Information Technology,
University of Nottingham, Nottingham NG8 1BB, UK

Abstract This paper investigates a tabu-search-based hyper-heuristic for solving examination timetabling problems. The hyper-heuristic framework uses a tabu list to monitor the performance of a collection of low-level heuristics and then make tabu heuristics that have been applied too many times, thus allowing other heuristics to be applied. Experiments carried out on examination timetabling datasets from the literature show that this approach is able to produce good quality solutions.

Keywords: hyper-heuristic, examination timetabling, heuristics, tabu search.

1. INTRODUCTION

This paper investigates a hyper-heuristic, based on tabu search, and its application to examination scheduling. The objective is to design a generic system that is able to select the most appropriate algorithm for the current instance of a given timetabling problem. Carter (1986), Carter and Laporte (1996) and Schaerf (1999) have conducted comprehensive surveys on various methods and strategies applied by researchers to solve timetabling problems. Many of these methods have successfully solved given problems and some algorithms/heuristics were reported to work well with particular data sets whilst others performed better when presented with different data sets. This indicates that one of the potential research issues in timetabling is to design a high-level algorithm that automatically, and intelligently, chooses a method suitable for a given problem instance (Burke and Petrovic, 2002).

This paper will report on our research into the design of a new hyper-heuristic framework using a tabu list and adaptive memory with the intention of monitoring and learning the behaviour and performance of low-level heuristics so as to help in making a well-informed decision of applying the best heuris-

tics at each decision point. We test our approach on examination timetabling problem using examination timetabling problem dataset publicly available at ftp://ftp.mie.utoronto.ca/pub/carter/testprob/.

The next section reviews the use of hyper-heuristic methodologies and tabu search related to timetabling problems. Section 2 gives a description of the examination timetabling problem. We describe our hyper-heuristic framework and strategy in Section 3 and Section 4 gives our experimental results and analysis. Section 5 concludes with a summary and presents future research directions.

1.1 Hyper-heuristics

The term hyper-heuristic (Burke *et al.*, 2003b) denotes a method that operates at a higher level of abstraction and can be thought of as a (meta-)heuristic that is able to intelligently choose a possible heuristic to be applied at any given time. We refer to Burke *et al.* (2003b) for further motivation and discussion on the emergence of the hyper-heuristic to solve optimisation problems. This includes references to earlier work that can be categorised as hyper-heuristic approaches, although they do not use this term.

One example of solving a large-scale university examination timetable problem using a hyper-heuristic approach can be seen in Terashima-Marin *et al.* (1999). Their approach had two phases in the construction of a timetable. Each phase used a different set of heuristics and a switch condition determined when to move from one phase to the other. A genetic algorithm, using a non-direct chromosome representation, was used to evolve the choice of heuristics, switch condition and strategies.

Burke and Newall (2004) proposed an adaptive method in constructing initial solutions for the examination timetabling problem. An initial ordering heuristic produced an order of exams to be scheduled. The ordering heuristic provides a good solution if the order is ideal, otherwise, it will adapt and improve the order, thus improving the initial solution. The results showed that the method could substantially improve the solution quality over the original heuristic (flat ordering, largest degree and smallest degree).

Cowling *et al.* (2001) use a choice function in their hyper-heuristic to determine which low level heuristic will be called next. The choice function adaptively ranks the low-level heuristics by considering recent improvement of each low-level heuristic, recent improvement of consecutive pairs of low-level heuristics and the number of CPU seconds elapsed since a particular heuristic was last called. The method was successfully tested on different applications: sales summit scheduling (Cowling *et al.*, 2001), nurse scheduling (Cowling *et al.*, 2002c) and project presentation scheduling (Cowling *et al.*, 2002b; Kendall *et al.*, 2002).

Cowling *et al.* (2002a) use a genetic algorithm based hyper-heuristic (Hyper-GA) to construct a sequence of heuristics that are applied to a trainer scheduling problem.

Nareyek (2001) proposed a learning procedure in a search process that learns to select promising heuristics based on weight adaptation. Their empirical study was carried out on two problems: Orc Quest and Logistics Domain.

Burke *et al.* (2003a) have used a tabu search hyper-heuristic (although different to the one proposed in this paper) and have successfully applied it to course timetabling and rostering problems. They used a ranking mechanism to dynamically rank each low-level heuristics. The heuristic with the highest rank will be applied in the next iteration and if the heuristic does not improve the solution, it will be placed in a tabu list. This tabu list is used to prevent non-performing heuristics from being chosen again in the near future. Our hyper-heuristic differs with respect to how we use the tabu list and how we choose and apply heuristic. Further details are given in Section 3.

There are other papers published on methods that are similar to the concept used in hyper-heuristics. It is not our intention to mention all of them, but nevertheless, it would be interesting to carry out a complete survey and categorise all papers that exhibit hyper-heuristic behaviour. From what we have seen from existing papers on hyper-heuristics, we believe that further research should be carried out in order to inject intelligence into the hyper-heuristic that does not depend on domain knowledge.

1.2 Tabu Search (Timetabling)

In this section, we will discuss briefly how other researchers apply tabu search approaches in solving timetabling problem. The basic form of tabu search (TS) is an idea proposed by Glover (1986) to solve combinatorial optimisation problems. The following is a definition by Glover and Laguna (1997):

> Tabu search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality.

The basic concept of tabu search is an extension of steepest descent by incorporating adaptive memory and responsive exploration. It uses memory not only to keep track of the current best solution but it also stores information related to the exploration process. Starting from the initial solution S_0, the algorithm iteratively explores a subset $N'(s)$ of the neighbourhood, $N(s)$, of the current solution s. The member with the lowest (assuming minimisation) value becomes the current solution irrespective of whether its value is better or worse than the current solution. Accepting a non-improving move will allow the search to continue to explore areas beyond local optima. However, this will typically lead to cycling, that is, repeatedly moving between some small set of solutions. To avoid this, it uses memory to store a tabu list. This list contains

moves that satisfy some *tabu restriction* criteria and these moves are prohibited for a predetermined number of iterations (*tabu tenure*). Moves that are in the tabu list are said to have a *tabu-active status*. An *aspiration criteria* is used to make a solution tabu free if the resultant evaluation is of sufficient quality and can prevent cycling.

Glover and Laguna (1997) also describe two important strategies used in tabu search: *intensification* and *diversification*. *Intensification* strategies involve changing the choice rules to intensify the search to examine neighbours of elite solutions. The idea is that if certain regions contained good solutions in the past they may possibly yield better solutions in the future. The *diversification* stage encourages the search process to examine unvisited regions and to generate solutions that differ significantly.

Schaerf and Schaerf (1995) apply tabu search techniques in scheduling lectures to periods for a large high-school. They represented their timetable as an integer-valued matrix M_{mxp} such that each row j of M represents the weekly assignment for teacher t_j. The type of moves used are atomic: moving a lecture to another period, and double moves which are moves made by a pair of atomic moves. The algorithm used a tabu search with atomic moves interleaved with a randomised non-ascendant method (RNA) using double moves. The RNA is used to generate the initial solution and is applied again after TS has given no improvements for a given number of iterations. The cycle is repeated allowing TS to start in a different direction. The tabu list is of variable size. Each move is inserted into the tabu list with a number of iterations I selected at random within a predetermined range. The *tabu tenure* therefore varies for each move. Each time a move is inserted the value I of all moves in the list will be decremented and once it reaches zero the move is removed. The algorithm uses the simplest *aspiration criterion* of accepting a tabu move only if it improves the current best solution. The algorithm gave good results for schools of various types, and for different settings of the weights of the objective functions. The timetable produced is better than the manual timetable and it was able to schedule 90–95% of the lectures.

Di Gaspero and Schaerf (2001) continued this research using tabu search for the examination timetabling problem. They modified their objective function using a shifting penalty mechanism (varying weights on soft and hard constraints) thus causing the search to explore different solution spaces. In order to decide which exams are to be moved, they maintain two violation lists: list of exams that violate either hard or soft constraints and list of exams that violate hard constraints only. During the search, they experiment on various strategies using shifting penalty mechanisms and the two violation lists. These two features, combined with a variable-size tabu list and starting the search with a good initial solution, were found to be helpful in directing the search into promising regions.

Di Gaspero (2002) and Di Gaspero and Schaerf (2003) further enhanced their algorithm by employing a multi-neighbourhood strategy applied to examination timetabling and course scheduling respectively. In the examination timetabling problem, Di Gaspero (2002) applied a combination of tabu search with different neighbourhoods (union and composition). He categorised these combinations into local search that specialised in optimising objective function (*recolour*), perturbing current solution (*shake*) or obtaining more improvement (*kick*). The *recolour* and *shake* algorithms were applied in sequence until there was no further improvement and the algorithm ended with the *kick*. The final results on seven benchmark datasets were better compared to the basic tabu search with single neighbourhood.

Thomson and Dowsland (1998) showed that it is possible to design robust solutions based on simulated annealing and tabu search by applying the algorithms on different case studies of scheduling, timetabling and staff-rostering problems in the education and hospital sectors. They apply varying *tabu restriction* on different moves and use a frequency-based *diversification* mechanism and penalised attributes that occur very frequently. Some of the modifications included can improve the tabu search but the implementation frequently depends on the precise details of the problem. Some of these modifications are different cost functions, variable tabu length list, combining moves into chains, strategic oscillation that force the search into different regions and prominent candidate list strategies.

White and Xie (2001) called their algorithm OTTABU and used it to provide an examination timetable using data provided by the University of Ottawa. The problem is modelled as a graph. The initial solution was generated using an algorithm derived from bin packing algorithms (largest enrolment first). The initial solution does not guarantee a feasible solution. A new solution is obtained by an atomic move. Their system used recency based short-term memory (TS) and frequency based long-term memory (TL) to improve the solution quality. The tenure of the short-term tabu list is found not to be critical if both longer term and short-term memory are used. Their experiments showed that longer term memory produced better schedules and since the longer term memory list can reduce its effectiveness, a quantitative analysis method is used to estimate the appropriate length of the longer term tabu list and a controlled tabu relaxation technique (emptying entries in TS and TL) is used to diversify the search. White *et al.* (2004) expand their research to include comparisons between their results and other published algorithms.

Wright (2001) incorporated sub-cost-guided search in both simulated annealing and tabu threshold acceptance methods. In tabu thresholding, the intensification and diversification are explicitly divided into two separate phases—the improving (intensifying) phase and the mixed (diversifying) phase. He used a focus form of diversification by accepting a solution even though the

overall cost had increased but one of the sub-costs had decreased. He experimented on modified school timetabling data and found significantly improved results.

The tabu search meta-heuristic has been explored in detail and applied to the examination timetabling problem by the above researchers. The main issues that were addressed and can be explored further are as follows:

- How can we use memory to help in storing history of previous moves (adaptive, short-term, long-term etc.)?
- What items should be stored in the tabu list?
- Neighbourhood size.
- Type of moves that dictate the next neighbour of a solution state.
- How to balance and decide when to intensify and diversify the search?
- Conditions for tabu restriction.
- Factors that affect tabu tenure?
- What aspiration criteria can be used to avoid missing a potentially good solution?

We incorporate some of the above issues into our hyper-heuristic framework and apply it to the examination timetabling problem.

2. PROBLEM DESCRIPTION

Timetabling is a special case of a scheduling problem (Wren, 1996). The layman's term for a timetable is normally used in an academic environment, which refers to a class timetable or examination timetable. A timetable normally tells you when and where events are to take place. Carter and Laporte (1996) defined the basic problem in examination timetabling as "the assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes". In some cases conflict cannot be avoided and the objective is to minimise the number of student conflicts.

We can represent the examination timetabling problem using a mathematical model. From the problem definition we know that it is an assignment type problem because we need to assign examinations to slots while minimising an objective function and satisfying a set of constraints. Thus we can formulate the problem as follows:

- E: a set of m examinations $E_1, E_2, \ldots, E_m$;
- S: a set of n slots $S_1, S_2, \ldots, S_n$;

- A final exam timetable T_{mn} such that $T_{ik} = 1$ if exam i is scheduled in slot k, 0 otherwise;
- A conflict matrix C_{mm} such that C_{ij} = total number of students sitting for both exams i and j;
- P_{ik} is a penalty given if exam i is scheduled in slot k.

The examination timetabling problem is to assign examinations to slots subject to some hard constraints that must be satisfied, and minimise soft constraint violation.

Hard constraints that must be satisfied are:

1 *Feasible.* The timetable must be feasible such that all exams must be scheduled and each exam $(E_1, E_2, \ldots, E_m)$ must be scheduled only once:

$$\sum_{k=1}^{n} T_{ik} = 1 \qquad i = 1, \ldots, m$$

2 *Student conflict.* No student should sit for more than one exam in the same slot. If exam i and exam j are scheduled in slot k, the number of students sitting for both exam i and $j(C_{ij})$ must be equal to zero, and this should be true for all exams already allocated:

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{n} T_{ik} T_{jk} C_{ij} = 0$$

We determine the quality of an examination timetable solution based on the penalty given if certain soft constraints are violated. The soft constraint that we would like to consider is the proximity constraint and a proximity cost is given when the proximity constraint is violated. A weighted proximity cost x_s is given whenever a student has to sit for two examinations scheduled s periods apart: these weights are $x_1 = 16$, $x_2 = 8$, $x_3 = 4$, $x_4 = 2$ and $x_5 = 1$.

P_{ik}, the total proximity cost if exam i is scheduled in slot k, is as follows:

$$P_{ik} = \sum_{j=1}^{m} X_{|k-l|} C_{ij}$$

where j (an exam in conflict with exam i) is scheduled in slot l.

Finally, our objective is to minimise the total proximity cost:

$$\sum_{i=1}^{m} \sum_{k=1}^{n} T_{ik} P_{ik}$$

Other additional soft constraints that are specific to university requirements can be added to this problem. But in this paper, we apply the same method of evaluating solution quality so that we can compare our results with other results published in the literature.

3. HYPER-HEURISTIC FRAMEWORK AND STRATEGY

A hyper-heuristic framework (Burke *et al.* 2003a,b) works at a higher level of abstraction than current (meta-)heuristic approaches and does not require domain knowledge. It only has access to non-domain-specific information that it receives from the heuristics that it operates upon. The hyper-heuristic can be implemented as a generic module that has a common interface to the various low-level heuristics and other domain-specific knowledge (typically the evaluation function) of the problem being solved. Initially, the hyper-heuristic needs to know the number of n heuristics provided by the low-level heuristic module. It will guide the search for good quality solutions by setting up its own strategy of calling and evaluating the performance of each heuristic known by their generic name $H_1, H_2, \ldots, H_n$. The hyper-heuristic does not need to know the name, purpose or implementation detail of each low-level heuristic. It just needs to call a specific heuristic, H_i, and the heuristic may modify the solution state and return the result via an evaluation function. The low-level-heuristic module can be viewed as a "black box" that hides the implementation detail and only returns a value.

3.1 Hyper-heuristic Module

The hyper-heuristic module is the main part of the research area where we need to design and test strategies that can intelligently select the best heuristic that will help guide the search to either intensify or diversify the exploration of the search region.

The general framework for our hyper-heuristic algorithm is as follows:

Step 1. Construct initial solution

Step 2. Do

Consider heuristics that are not tabu

Apply chosen heuristic and make the heuristic tabu

Update solution

Until terminating condition

The initial solution is produced using a constructive heuristic (largest degree or saturation degree (Carter and Laporte, 1996). The initial solution need

not be a good solution and it may not be feasible (i.e. some exams are unscheduled). The algorithm works with infeasible solutions since some of the low-level heuristics specialised in scheduling unscheduled exams. Next, a randomisation (randomly move exams to other valid slots) is carried out to start different runs with different solutions. In Step 2 we explore the neighbourhood to search for a better solution or local optima (and possibly global optima). The framework is similar to a local search except that in Step 2 we explore the neighbourhood by selecting which heuristic to use to update the current solution.

Our hyper-heuristic differs from other neighbourhood search algorithms or meta-heuristics (such as Tabu Search and Simulated Annealing) with respect to the management of several heuristics. The hyper-heuristic manages the heuristics by selecting which heuristic(s) should be considered and which heuristic(s) should be applied. In fact, the heuristics being considered can be a local search algorithm or just a simple move operator.

The hyper-heuristic is like a manager who employs a team of heuristic workers. A good manager does not need to know how the workers do their job but it must be intelligent in recognising when a good job is done. The workers may be good or poor and sometimes a good combination of team workers will produce good solution. Normally when we have a team of workers, rather than asking them to work in sequence, we can ask them to perform their specific task simultaneously and whoever produce the best work will be accepted. Their progress will be monitored so that the manager can learn and recognise each workers' specialisation and will be able to decide and select the next team of workers.

Therefore, we can view the hyper-heuristic as a manager and the collection of heuristics as a team of workers who are given an area in the solution space (the heuristic search space, which is part of the solution search space) and their task is to find a good solution and return it. The heuristic may be doing a complex task by intensively exploring a large neighbourhood search space or it may just do a simple task of exploiting a very small neighbourhood of solutions. In the search for good quality solutions, the hyper-heuristic exhibits a kind of reinforcement learning, which will assist in an intelligent action at each decision point. It monitors the behaviour of each low-level heuristic by storing information about the performance using adaptive memory. Our hyper-heuristic uses a tabu list that is of a fixed length n, where n is the number of low-level heuristics. Instead of storing moves, each tabu entry stores (non-domain) information about each heuristic i.e., heuristic number, recent change in evaluation function, CPU time taken to run the heuristic, and tabu status (or tabu duration, as the term we prefer to use). Tabu duration indicates how long a heuristic should remain tabu (0–4) and therefore not be applied in the current iteration. If the tabu duration is zero, the heuristic is said to be tabu inactive

and can be applied to update the solution. If the tabu duration is non-zero, the heuristic is said to be tabu active and may not be used to update the solution. A heuristic is made tabu when it satisfies our tabu restriction conditions. We do not use any aspiration criteria (changing a tabu active status to tabu inactive because the heuristic improves the solution) at this point because we wanted to compare which tabu duration produces the best quality solution. Therefore, the only time a heuristic changes its status from tabu active to tabu inactive is when the tabu duration is zero. The tabu duration is set for a heuristic whenever a tabu restriction is satisfied. After each iteration, the tabu duration is decremented until it reaches zero and the heuristic is now tabu inactive. For each test on the dataset we fixed the tabu duration at between zero and four and, in this paper, we compare to find which tabu duration produces better quality solutions.

We use several strategies when considering the heuristics: consider all heuristics (i.e. no tabu criteria), consider heuristics that are not tabu, or consider heuristics that lead to improvement only. Each heuristic differs in how it decides to move, thus creating its own search space region (heuristic search space) in the solution search space. At each choice point, we need to decide whether we want to intensify the search in a particular region by applying the same heuristic or to diversify the search into another region by applying a different heuristic. At this point, the hyper-heuristic can actually choose intelligently when to intensify or diversify because we believe that allowing the low-level heuristics to compete at each iteration and selecting the heuristic with the best performance will help to balance the diversification and intensification of the solution search space. Heuristics that have been applied become tabu so that in the next iteration we can look at the possibility of other low-level heuristics that may perform well, but perhaps not as well as the previous heuristics that are now tabu active. We have implemented the simplest strategy, i.e. hyper-heuristic with fixed tabu duration (HH-FTD), where we consider all tabu inactive heuristics and apply the heuristic that has the best improvement only. The algorithm iterates for a fixed time or until there is no further improvement for a given number of heuristic calls.

3.2 Low-Level Heuristics Module

Low-level heuristics are heuristics that allow movement through a solution space that require domain knowledge and are problem dependent. Each heuristic creates its own heuristic search space that is part of the solution search space. The idea is to build a collection of possible simple moves or choices since we would like to provide a library of heuristics that can be selected intelligently by a hyper-heuristic tool. This library, at the moment, will only include simple low-level heuristics and future work will include the possibility

of adding other meta-heuristics such as Simulated Annealing, Tabu Search or a Memetic Algorithm.

The heuristics change the current state of a problem into a new state by accepting a current solution and returning a new solution. Each low-level heuristic can be considered as an improvement heuristic that returns a move, a change in the penalty function and the amount of time taken to execute the heuristic. The best performing heuristic should cause a maximum decrease in penalty (the lowest value). Each move from an individual heuristic may cause the search to probe into the current neighbourhood or to explore into a different neighbourhood. A change in the penalty value means changing the penalty value for each of the soft constraints that were violated (first-order conflict, second-order conflict, etc) or moving an exam into an unscheduled list (exam becomes unscheduled and violates hard constraints).

We have implemented the following low-level heuristics, grouped into four categories:

1 Select and schedule exam. Selecting the next exam to schedule is dependent upon which factor is considered to be important in determining the difficulty of scheduling an exam. The strategies that have been used in the literature, and that are adapted from the graph colouring heuristics, are used here. Once an exam is selected, it will be scheduled into the best available slot that will maximise the reduction in penalty.

   - Largest enrolment: exam with largest enrolment should be selected since it might be difficult to schedule at a later time.
   - Largest exam conflict: exam that is in conflict with the largest number of exams is normally considered to be more difficult to schedule.
   - Largest total student conflict: exam that has maximum total number of students in conflict.
   - Largest exam conflict already scheduled: exam that has the greatest number of exams in conflict already scheduled would be difficult to schedule since it would have less choice of valid slots.
   - Exam with least valid slots: exam that has the least valid slots should be scheduled now since it may not have any slots available at a later stage.

2 Move exam i from location x to y.

   - Select an exam at random and move to another random slot.
   - Move exam i with maximum penalty from randomly selected exams.

- Move exam i with highest second-order conflict from location x to a new location y.
- Move exam i with highest second order conflict from location x to a new location y which maximises the reduction in second order conflict.
- Move exam i with first order conflict from location x to a new location y which does not result in first-order conflict.

3 Swap.

- Random: select an exam at random and find another exam at random which can swap slots.
- Min–max swap: swap the slots for exam with minimum penalty and exam with maximum penalty.

4 Remove. Remove a randomly selected exam from the examinations already scheduled.

All of the above low-level heuristics are either 1-opt or 2-opt and there is also a mixture of some randomness and deterministic selection of exams and slots. We purposely test low-level heuristics with simple moves rather than low-level heuristic with intelligence and complex moves because we want to make sure that the hyper-heuristic can recognise good moves and make an intelligent decision based on these simple moves. Furthermore, we want to make the problem-domain knowledge heuristics easy to implement and the hyper-heuristic more generalised.

4. EXPERIMENTAL RESULTS

We have implemented and tested our tabu-search-based hyper-heuristic (TSHH) framework on a PC with an AMD Athlon 1 GHz processor, 128 Mb RAM and Windows 2000. The program was coded in C++ using an object-oriented approach. We defined and implemented the hyper-heuristic and heuristics as objects that have a common interface and can interact with each other. Once the hyper-heuristic object is fully defined, implemented and tested with a set of heuristics object for one application, we can easily reuse the hyper-heuristic object with another set of heuristic objects for a different application. This approach should be cost effective because it can reduce the complexity of building another system. Thus, we can easily produce solutions to users who require "good enough–soon enough–cheap enough" (Burke *et al.* 2003a, 2003b) solutions to their problems by implementing several domain specific low-level heuristics with simple moves.

Therefore, the objectives of our experiments are:

Table 1. Characteristics of real problems.

Code	Institution	No. of slots	No. of exams	No. of students exams	No. of student density	Conflict matrix
Car-f92	Carleton University, Ottawa	32	543	18,419	55,522	13.8%
Car-s91	Carleton University, Ottawa	35	682	16,925	56,877	12.8%
Ear-f83	Earl Haig Collegiate Institute, Toronto	24	189	1,125	8,109	26.7%
Hec-s92	Ecoles des Hautes Etudes Commercials, Montreal	18	81	2,823	10,632	42.0%
Kfu-s93	King Fahd University Of Petroleum and Minerals, Dharan	20	461	5,349	25,113	5.6%
Sta-f83	St Andrew's Junior High School	13	139	611	5,751	14.4%
Tre-s92	Trent University, Peterborough, Toronto	23	261	4,360	14,901	5.8%
Ute-s92	Faculty of Engineering, University of Toronto	10	184	2,750	11,793	8.5%

- To establish a well defined interface between our hyper-heuristic module and our low-level heuristics module;
- To compare the quality of results produced by our hyper-heuristic with other known methods published using similar quality measures;
- To demonstrate that the hyper-heuristic module does not need to rely upon domain knowledge to make its decisions;
- To demonstrate that the hyper-heuristic can manage and choose the low-level heuristics at each decision point in a search;
- To evaluate the performance of low-level heuristic;
- To determine further improvement in our hyper-heuristic module.

4.1 Datasets

We tested our implementation with datasets taken from established datasets made public and used by a number of other examination timetabling researchers. The datasets were provided by Michael Carter and can be downloaded from

Table 2. Results with 10 minute and 4 hour run.

File	Hyper-heuristic with fixed TD (best penalty value per student from 8 runs)					HH-FTD (TD = 2, 4 h run)	% improve with long run
	TD = 0	TD = 1	TD = 2	TD = 3	TD = 4		
Car-f92	5.94	5.52	**5.46**	5.63	6.02	**4.67**	14.47%
Car-s91	6.91	6.47	**6.32**	6.98	7.10	**5.37**	15.03%
Ear-f83	47.01	44.58	43.58	**43.54**	45.16	**40.18**	7.80%
Hec-s92	13.84	14.09	12.79	**12.24**	12.86	**11.86**	7.27%
Kfu-s93	20.53	18.32	**18.08**	19.22	19.86	**15.84**	12.39%
Sta-f83	168.4	165.8	**165.6**	166.3	167.1	**157.38**	4.96%
Tre-s92	11.01	10.99	**9.79**	10.99	10.59	**8.39**	14.30%
Ute-s92	29.34	28.18	**27.97**	29.20	31.05	**27.60**	1.32%

ftp://ftp.mie.utoronto.ca/pub/carter/testprob/. Table 1 shows the characteristics of each dataset. Each datasets is stored in two files; one file contains a list of courses and their enrolment, and the other a list of student and their course selections. We test our method on eight of the datasets. We use both data files to construct a conflict graph and the largest degree algorithm (Carter *et al.* 1996) to construct the initial solution. The density of the conflict matrix in Table 1 is calculated as the average number of other exams that each exam conflicts with, divided by the total number of exams. For example, a conflict matrix density of 0.5 or 50% indicates that each exam conflicts with half of the other exams on average.

The numbers of slots are obtained from results reported by Carter *et al.* (1996). They used five different graph colouring heuristics to determine the minimum number of slots (sessions) required to produce a feasible solution subject to a no-clash constraint.

4.2 Experimental Results and Analysis

Our hyper-heuristic, with fixed tabu duration (HH-FTD), was tested with eight benchmark datasets. For each dataset, we experimented with tabu durations varying from 0 to 4 and with two different terminating conditions (no further improvement for the last 10,000 iterations or running time of 10 minutes). In Table 2, the first column shows the file name of each dataset and the next five columns show our results (best penalty value per student) with a tabu duration varying from 0 to 4. We do not show here the actual time that it finds the best solution but the best results are normally found towards the end of the search. After further analysis on the performance graph we found that improvements are still being made toward the end of run time. Therefore, we run the algorithm again for four hours with a tabu duration of 2 (many of the

datasets work best with tabu duration of 2), to see whether much better solutions can be found if we run the algorithm longer than the 10 minutes that we used previously. The last column in Table 2 shows that prolonging the algorithm does improve the solution further and it demonstrates that it is robust and able to avoid being trapped in local optima.

When the tabu duration is 0, the hyper-heuristic does not make any heuristics tabu and, since none of the results is the best among the datasets, we can conclude that we do need to use a tabu list to guide the hyper-heuristic in its heuristic selection. In our tabu-based hyper-heuristic strategy, we apply the concept of heuristics cooperating with each other rather than penalising a non-performing heuristic. When TD is greater than zero, we apply a tabu restriction where a heuristic will be tabu active if its solution value has been accepted to update the current solution. The heuristic will remain tabu active for a number of steps equal to TD. We made a heuristic that has been applied tabu because we want to direct the search to other possible heuristic search spaces. Eventually we may go back to a heuristic search space once it is no longer tabu active and can give the best solution amongst all tabu inactive heuristics. The best result for six of the datasets is when TD is two and for two of the datasets, the best result is when TD = 3. It is interesting to find that two datasets (Ear-f83 and Hec-s92) obtain best result when TD is higher and has a higher conflict matrix density (see Table 1), i.e. 26.7% and 42.0%. An examination timetabling dataset with higher conflict matrix density would imply that we might have less and sparsely distributed solution points (feasible solution) in our solution space since too many exams are conflicting with each other. Thus, a higher TD may force it to diversify its exploration of the solution search space by allowing it to move from one heuristic search space to another. For each of the datasets, except one dataset (Hec-s92), the average penalty started to decrease as we increased the TD and began to increase again once it reached its minimum average penalty. This shows that the hyper-heuristic does need to decide which TD is best for each dataset because a tabu duration which is too high or too low will produce worse solutions.

Figure 1 shows the hyper-heuristic performance with different TD on car-f92 dataset. This dataset is one of the largest dataset with 543 exams to schedule in 32 slots and with a total number of students of 18,419. This graph demonstrates how the hyper-heuristic explores the search space. The x-axis represents the iteration steps up to 250,000 moves while the y-axis represents overall penalty cost. Note that the timetable quality is measured by taking the average penalty per student. The curve shows that the algorithm begins with an initial solution and rapidly improves the result in less than 10,000 moves. The graph shows fluctuations because at every move we accept a solution from the best performance heuristic even though it does not improve the solution. The higher TD means that the heuristics will remain tabu longer,

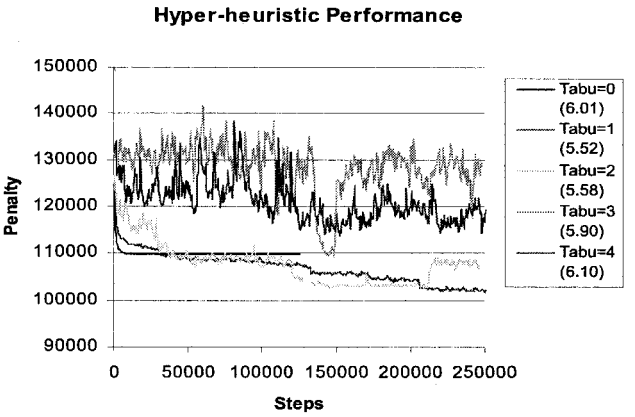


Figure 1. Hyper-heuristic performance with different TD.

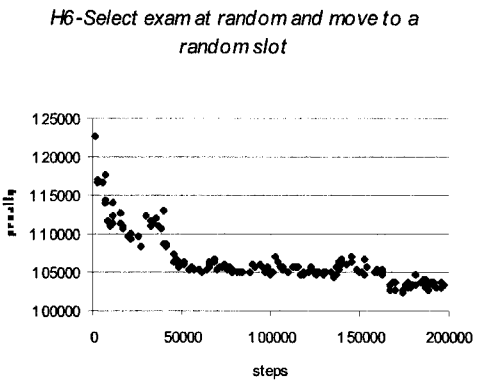


Figure 2. Heuristic 6.

thus allowing other heuristics to be applied next. By increasing the TD, we notice that the next solution accepted may make the solution much worse but it can still improve the solution in the next move. So, a tabu duration value does help to improve solution quality, and too high a value may make the solution much worse, making it difficult to improve it again. The simplest form of this hyper-heuristic does not limit the range of how much a worse solution may be accepted but further investigation on this hyper-heuristic will limit the acceptance of worst solution.

The graphs in Figures 2 and 3 show when two of the heuristics were applied and how the two heuristics change the solution state.

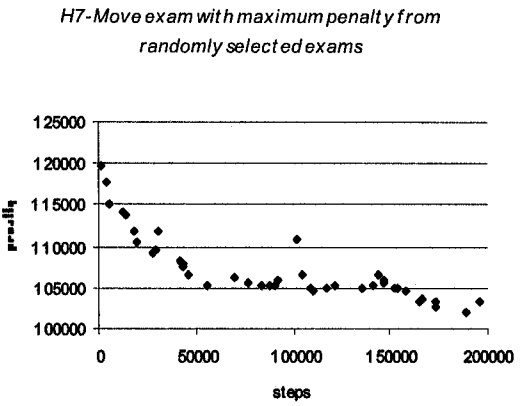


Figure 3. Heuristic 7.

The larger number of plots in Figure 2 compared to Figure 3 indicates that heuristic 6 has been applied more than heuristic 7. We do not show graphs for other heuristics because the shape is almost the same except that some are applied more than others. We also do not show how each heuristic performs in all iterations because we only keep track of its performance when it is applied. We can also see that for different datasets certain heuristics will be applied more than others, therefore it is justifiable to use several low-level heuristics that can compete with each other and a hyper-heuristic can than select the best low-level heuristic to be applied, given not only the point in the search space but also a specific problem instance.

Table 3 shows our four hour run results compared to other published results for benchmark datasets. Our objective here is to show that the HH-FTD is able to produce good quality and feasible solutions for examination timetabling problems even though it may not produce the best results. The results show that our generic method is able to produce good quality solutions compared to the others. The first two that we compare results against are of Di Gaspero and Schaerf (2001) who use tabu search and Di Gaspero (2002) who use tabu search with multi-neighbourhood. Our results are better than the tabu search method in all cases and almost as good as the tabu search with multi-neighbourhood. We also compare our results with results from other methods: constructive heuristics with backtracking of Carter *et al.* (1996); the memetic algorithm of Burke and Newall (1999); the greedy constructive heuristic with an optimiser by Caramia *et al.* (2001); and a hybrid of constraint programming, simulated annealing and hill climbing with Kempe chain neighbourhood by Merlot *et al.* (2003). Our results are better than Carter *et al.* (1996) and

Table 3. Comparing our best results and published results.

File	HH-FTD (TD = 2, 4 h run)	Di Gaspero and Schaerf	Di Gaspero	Carter *et al.*	Caramia *et al.*	Merlot *et al.*	Burke and Newall
Car-f92	4.67	5.2	-	6.2–7.6	6.0	4.3	**4.2**
Car-s91	5.37	6.2	5.68	7.1–7.9	6.6	5.1	**4.8**
Ear-f83	40.18	45.7	39.36	36.4-46.5	**29.3**	35.1	35.4
Hec-s92	11.86	12.4	10.91	10.8–15.9	**9.2**	10.6	10.8
Kfu-s93	15.84	18.0	-	14.0–20.8	13.8	**13.5**	13.7
Sta-f83	157.38	160.8	157.43	161.5–165.7	158.2	**157.3**	159.1
Tre-s92	8.39	10.0	-	9.6–11.0	9.4	8.4	**8.3**
Ute-s92	27.60	29.0	-	25.8–38.3	**24.4**	25.1	25.7

Caramia *et al.* (2001) in four cases. In all cases we could not produce better results than Burke and Newall (1999) and Merlot *et al.* (2003).

As a whole, we can see that our method does not perform the worst or best in any cases, but works reasonably well across all problem instances. We believe that with further enhancements in our hyper-heuristic selection method and some adaptive tabu duration, we can improve our results.

5. CONCLUSIONS AND FUTURE WORK

The simplest form of the hyper-heuristic module HH-FTD has been implemented and tested on exam timetabling benchmark data. Preliminary results showed that it is not able to beat the best results in the literature but it is able to produce good quality solutions. Our objective is not to beat the best solution but to show that the hyper-heuristic module produces good solutions that are feasible and will work across all problem instances and other real-world problems. Our generic solution methodology can easily be applied to other problems by just changing the low-level heuristics and the evaluation function while the search method remains the same.

Currently, we are testing a more advanced hyper-heuristic module that includes more tabu criteria such as tabu criteria based on CPU time, tabu based on change in penalty function and a probabilistic heuristic selection. In the future, we will experiment on adaptive tabu strategies and apply our method on a larger timetabling instance as well as other applications.

References

Burke. E. K., Kendall, G. and Soubeiga, E. (2003a) A tabu search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, **9**:451–470.

Burke, E. K. and Petrovic, S. (2002), Recent research directions in automated timetabling. *European Journal of Operational Research*, **140**:266–280.

Burke, E. K. and Newall, J. P. (2004), Solving examination timetabling problems through adaptation of heuristics orderings. *Annals of Operations Research*, **129**:107–134.

Burke, E. K. and Newall, J. P. (1999) A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, **3**:63–74.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S. (2003b) Hyper-Heuristics: An emerging direction in modern search technology. In *Handbook of Meta-Heuristics*, F. Glover and G. Kochenberger (Eds.), Chapter 16, Kluwer, Dordrecht, pp. 457–474.

Caramia, M., Dell'Olmo, P. and Italiano, G. F. (2001) New algorithms for examination timetabling. In *Proceedings of 4th Workshop on Algorithm Engineering*, Lecture Notes in Computer Science, Vol. 1982, Springer, Berlin, pp. 230–242.

Carter, M. W. and Laporte, G. (1996) Recent developments in practical examination timetabling. *The Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1153, E. K. Burke and P. Ross (Eds.), Springer, Berlin, pp. 3–21.

Carter, M. W. (1986) A Survey of practical applications of examination timetabling algorithms. *Operations Research Society of America*, **34**:2, March–April.

Carter, M. W., Laporte, G. and Lee, S. Y. (1996) Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, **47**:373–383.

Cowling, P., Kendall, G. and Soubeiga, E. (2001) A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, Lecture Notes in Computer Science, Vol. 2079, E. K. Burke and W. Erben, (Eds.), Springer, Berlin, pp. 176–190.

Cowling, P., Kendall, G. and Han, L. (2002a) An Investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of Congress on Evolutionary Computation (CEC2002)*, pp. 1185–1190.

Cowling, P., Kendall, G. and Soubeiga, E. (2002b) Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Applications of Evolutionary Computing: Proceedings of EVO Workshop 2002*, Lecture Notes in Computer Science, Vol. 2279, S. Cagoni, J. Gottlieb, E. Hart, M. Middendorf and R. Günther (Eds.), Springer, Berlin, pp 1–10.

Cowling, P., Kendall, G. and Soubeiga, E. (2002c) Hyperheuristics: A robust optimisation method applied to nurse scheduling. *7th International Conference on Parallel Problem Solving from Nature, PPSN2002*, Lecture Notes in Computer Science, Vol. 2439, Springer, Berlin, pp. 851–860.

Di Gaspero, L. (2002) Recolour, shake and kick: A recipe for the examination timetabling problem. In *Proceedings of the Fourth International Conference on the Practice and Theory of Automated Timetabling*, Gent, Belgium, August 2002, E. Burke and P. De Causmaecker (Eds.), pp. 404–407.

Di Gaspero L. and Schaerf A. (2001), Tabu search techniques for examination timetabling. In *Practice and Theory of Automated Timetabling III*, E. K. Burke and W. Erben (Eds.), Lecture Notes in Computer Science, Vol. 2079, Springer, Berlin, pp. 104–117.

Di Gaspero, L. and Schaerf, A. (2003) Multi-neighbourhood local search with application to course timetabling. In *Practice and Theory of Automated Timetabling IV*, E. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 262–275.

Downsland, K. (1998) Off-the-peg or made to measure: timetabling and scheduling with SA and TS. In *Practice and Theory of Automated Timetabling II*, E. Burke and M. Carter (Eds.), Lecture Notes in Computer Science, Vol. 1408, Springer, Berlin, pp. 37–52.

Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, **13**:533–549.

Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer, Boston, MA.

Gratch, J. M. and Chien, S. A. (1996) Adaptive problem-solving for large scale scheduling problems: A case study. *Journal of Artificial Intelligence Research*, **4**:365–396.

Kendall, G., Soubeiga, E. and Cowling, P. (2002) Choice function and random hyperheuristics. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning, SEAL'02*, pp. 667–671.

Merlot, L. T. G., Boland, N., Hughes, B. D. and Stuckey, P. J. (2003) A hybrid algorithm for the examination timetabling problem. In *Practice and Theory of Automated Timetabling IV*, E. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 207–231.

Nareyek, A. (2001) An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. *Proceedings of 4th Metaheuristics International Conference, MIC'2001*, pp. 211–216.

Schaerf, A. and Schaerf, M. (1995) Local search techniques for high school timetabling. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT'95)*, E. K. Burke and P. Ross (Eds.), pp. 313–323.

Schaerf, A. (1999) A survey of automated timetabling. *Artificial Intelligence Review*, **13**:87–127.

Terashima-Marín, H., Ross, P. M. and Valenzuela-Rendón, M. (1999) Clique-based crossover for solving the timetabling problem with gas. *Proceedings of the Congress on Evolutionary Computation*, Washington, DC, July 6–9, pp. 1200–1206.

White, G. M. and Xie, B. S. (2001) Examination timetables and tabu search with longer term memory. In *Practice and Theory of Automated Timetabling III*, E. K. Burke and W. Erben (Eds.), Lecture Notes in Computer Science, Vol. 2079, Springer, Berlin, pp. 85–103.

White, G. M., Xie, B. S. and Zonjic, S. (2004) Using tabu search with longer-term memory and relaxation to create examination timetables. *European Journal of Operational Research*, **153**:80–91.

Wren, A. (1996) Scheduling, Timetabling and Rostering—a special relationship? In *Practice and Theory of Automated Timetabling I*, E. K. Burke and P. Ross (Eds.), Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 46–76.

Wright, M. (2001) Subcost-guided search—experiments with timetabling problems. *Journal of Heuristics*, **7**:251–260.

Sports Scheduling

ROUND ROBIN TOURNAMENTS WITH ONE BYE AND NO BREAKS IN HOME–AWAY PATTERNS ARE UNIQUE

Dalibor Fronček
Department of Mathematics and Statistics, University of Minnesota Duluth
1117 University Drive, Duluth, MN 55812, USA
and Technical University Ostrava, Czech Republic
dfroncek@d.umn.edu

Mariusz Meszka
Faculty of Applied Mathematics, AGH University of Science and Technology
Mickiewicza 30, 30-059 Krakow, Poland
meszka@agh.edu.pl

Abstract We examine round robin tournaments with m teams and m rounds, for $m \geq 3$, with the property that every team plays no game in one round and exactly one game in each of the remaining $m-1$ rounds. We show that for every such m there exists a unique schedule in which no team plays two consecutive home or away games.

Keywords: scheduling tournaments, round robin tournament, home–away pattern, complete graph factorization.

1. INTRODUCTORY NOTES AND DEFINITIONS

Many sport competitions are played as round robin tournaments. A *round* is a collection of games in which every team plays at most one game. A *k-round round robin tournament of m teams*, denoted $RRT(m,k)$, is a tournament in which each team meets every other team exactly once and the games are divided into k rounds. A schedule, which is played in the minimum number of rounds possible is called *compact*; if more than the minimum number of rounds is used the schedule is *non-compact*. Although tournaments where every pair of opponents meets exactly l times (called *l-leg tournaments*) are very com-

mon, we will discuss only 1-leg tournaments here. An l-leg tournament can be indeed scheduled as a 1-leg tournament repeated l times with teams exchanging their respective home fields regularly. There are many different models that are widely used. In some competitions, like North-American NHL, NBA, NFL and others, the teams are divided into several divisions and it is required that games "inside" the divisions (called *intradivisional* games) and "across" the divisions (called *interdivisional* games) are distributed according to some rules. These rules often take into account travel distances. Therefore a team usually plays several games in a row with teams of another division at their fields. Then there follow several games played at the team's home field or with teams of the same division. Many other constraints are also considered. These can include TV schedules, availability of fields/stadiums, traditional rivals, etc. However, the schedule is usually not strictly divided into rounds and the number of days when the games are played is therefore larger than the necessary minimum. Construction of schedules of this kind is usually based on optimization methods like integer programming or finite-domain constraint programming (see e.g. Henz, 1999, 2001; Henz, *et al.*, 2003; Nemhauser and Trick, 1998; Schaerf, 1999; Schreuder, 1992; Trick, 2000). The result is then an exact schedule in which the dates and fields of all games between particular pairs of opponents are assigned. A graph-theoretic approach can be used for leagues with a small number of teams (see e.g. Dinitz and Froncek, 2000).

In other cases the rules are based on certain restrictions resulting from a limited number of available fields and/or suitable time slots. Schedules of this type were studied among others by Finizio (1993) and Straley (1983). On the other hand, most European national football (soccer) leagues are scheduled as 2-leg compact round robin tournaments (see e.g. Griggs and Rosa, 1996; UEFA, 2004). These tournaments are usually scheduled in such a way that a schedule for a 1-leg $RRT(2n, 2n-1)$ is repeated twice. It is then required that for each team the home and away games should interchange as regularly as possible provided that each team meets every opponent in one leg at its own field and in the other leg at the opponent's field.

In competitions that are played in regular rounds it is usually desirable that for each team the home games and away games interchange as regularly as possible. The leagues often have fixed *draw tables* (or *generic schedules*) with teams denoted just $1, 2, \ldots, m$ that are used repeatedly every season. The teams then draw their numbers either from the whole pool of m numbers (if they have no specific requirements) or from a limited pool (if they have some specific constraints). In this paper we actually present such generic schedules. Fundamental theoretical results concerning such generic schedules were studied by de Werra (1981) and Schreuder (1980).

2. SCHEDULES WITH ONE BYE

In what follows we consider $RRT(2n, 2n-1)$. The *home–away pattern* of a team i, denoted $HAP(i)$, is a sequence $a_1(i), a_2(i), \ldots, a_{2n-1}(i)$, where $a_j(i) = H$ if team i plays in round j a game in the home field and $a_j(i) = A$ if team i plays in round j a game in the opponent's field. If the regularity of the home–away patterns is our top priority, then the most desirable HAP is indeed either $AHAH \ldots AH$ or $HAHA \ldots HA$ in which no subsequence AA or HH appears. Obviously, one can never find a schedule in which all teams would have one of these two HAPs. In this case the teams, which start the season with a home game would never meet. A natural way to measure how "good" a given schedule is is to count the number of breaks in HAPs. A *break* in the HAP of team i is a subsequence AA or HH. Therefore, if we concentrate only on HAPs, we can say that the best schedule is the one with the least number of breaks. By a *break game* we mean the second game in any sequence AA or HH. For instance, in the sequence $HHHAA$ the break games are the games in rounds $2, 3, 5$. Two teams i_1 and i_2 have *complementary HAPs* if $a_j(i_1) = A$ if and only if $a_j(i_2) = H$.

The best possible schedule with respect to the number of breaks is given by the following theorem, which was proved by de Werra (1981).

Theorem 1 *In an $RRT(2n, 2n-1)$, the least number of breaks is $2n-2$. It can be attained in such a way that there are exactly $n-1$ teams with a home break, $n-1$ teams with an away break and 2 teams with no break. There are exactly $n-1$ rounds with break games, each of them containing exactly one home break game and one away break game.*

2.1 Odd Number of Teams

It is well known that a schedule for an odd number of teams, $2n-1$, can be constructed by taking a schedule for $2n$ teams and leaving out one team (called the *dummy team*). Then the team i that was scheduled to play the dummy team in round j plays no game in that round and is said to have a *bye*. We denote a bye in $HAP(i)$ by $a_j(i) = B$. It is also well known that the most commonly used schedule, sometimes called the *canonical* or *1-rotational* schedule has the nice property that if we let the dummy team be the team $2n$, then the remaining teams have no breaks in their schedules. This includes also no breaks around byes, that is, there is no sequence AA, HH, HBH, or ABA in any HAP. The schedule is described in the following construction.

Construction 2 We construct an $RRT(2n+1, 2n+1)$. First we introduce some necessary notation. When a game between teams i and k is scheduled for round j, we denote it by $g(i,k) = g(k,i) = j$. Set $g(i,k) = g(k,i) = i+k-1$. Obviously, $g(i,i) = 2i-1$ means that the team i has a bye in the round $2i-1$.

Table 1. $RRT(7,7)$.

R 1	R 2	R 3	R 4	R 5	R 6	R 7
$7-2$	$4-6$	$1-3$	$5-7$	$2-4$	$6-1$	$3-5$
$6-3$	$3-7$	$7-4$	$4-1$	$1-5$	$5-2$	$2-6$
$5-4$	$2-1$	$6-5$	$3-2$	$7-6$	$4-3$	$1-7$
1 bye	5 bye	2 bye	6 bye	3 bye	7 bye	4 bye

The addition is modulo $2n+1$ with the exception that 0 is replaced by $2n+1$. Home field is determined as follows. In the first round, team 1 has a bye, teams $2, 3, \ldots, n+1$ play home and teams $n+2, n+3, \ldots, 2n+1$ play away. We observe that having scheduled a round j, we can obtain opponents for round $j+1$ by adding $n+1$ to each team number. That is, if $j = g(i,k) = g(k,i) = i+k-1$ with i playing home and k away, then

$$g(i+n+1, k+n+1) = (i+n+1)+(k+n+1)-1 = i+k = j+1$$

and the team $(i+n+1)$ plays home while $(k+n+1)$ plays away.

An example of the schedule for seven teams is shown in Table 1. A game between teams i and k with i playing home is denoted by $k-i$.

Surprisingly, this schedule is the only one with this property. Notice that for schedules with byes the definition of complementary HAPs of teams i_1, i_2 requires the following: If $a_j(i_1) = B$ for some j, then also $a_j(i_2) = B$.

Theorem 3 *For every $n \geq 1$ there exists an $RRT(2n+1, 2n+1)$ such that no $HAP(i)$ contains any sequence AA, HH, HBH, or ABA. Moreover, for each such n, the schedule is unique up to permutation of team numbers.*

Proof. The existence was proved in Construction 2. Now we prove the uniqueness. First we observe that there is exactly one team with a bye in each round. In an $RRT(2n+1, 2n+1)$ we need to play $n(2n+1)$ games. Because we can schedule at most n games in each of the $2n+1$ rounds, it is easy to see that there must be *exactly* n games in each round.

As opposed to the notation used in Construction 2, we will assume that a team $i, i = 1, 2, \ldots, 2n+1$, has a bye in the round i. That is, $a_i(i) = B$. For clarity, we present in Table 2 the schedule for seven teams again following the notation used in this proof. We can observe that the schedule here can be obtained from Construction 2 by the permutation $\pi(i) = 2i-1$.

We can without loss of generality (WLOG) assume that $a_2(1) = H, a_3(1) = A$ and so on. Then, because $a_2(2) = B$ and teams 1 and 2 cannot play in either

Table 2. $RRT(7,7)$.

R 1	R 2	R 3	R 4	R 5	R 6	R 7
$6-3$	$7-4$	$1-5$	$2-6$	$3-7$	$4-1$	$5-2$
$4-5$	$5-6$	$6-7$	$7-1$	$1-2$	$2-3$	$3-4$
$2-7$	$3-1$	$4-2$	$5-3$	$6-4$	$7-5$	$1-6$
1 bye	2 bye	3 bye	4 bye	5 bye	6 bye	7 bye

Table 3. HAP for $RRT(2n+1, 2n+1)$.

	1	2	3	4	...	$2n-2$	$2n-1$	$2n$	$2n+1$
1	**B**	H	A	H	...	H	A	H	A
2	A	**B**	H	A	...	A	H	A	H
3	H	A	**B**	H	...	H	A	H	A
4	A	H	A	**B**	...	A	H	A	**H**
...					...				
...					...				
...					...				
$2n-2$	A	H	A	H	...	**B**	H	A	H
$2n-1$	H	A	H	A	...	A	**B**	H	A
$2n$	A	H	A	H	...	H	A	**B**	H
$2n+1$	H	A	H	A	...	A	H	A	**B**

round 1 or 2 since one of them has a bye in each of these rounds, we can see that $a_1(2) = A, a_3(2) = H, a_4(2) = A$ and so on. For similar reasons, because $a_3(3) = B$, we have $a_1(3) = H, a_2(3) = A, a_4(3) = H, \ldots$ or otherwise the teams 2 and 3 can never play against each other. Inductively, we can see that for teams i and $i+1$ one of them has to start the schedule with a home game while the other one with an away game otherwise they never meet. An example is shown in Table 3.

We introduce some more notation. By $S(i,k)$ we denote the set of all rounds in which teams i and k can possibly meet. In other words, $j \in S(i,k)$ if and only if $a_j(i) = H$ and $a_j(k) = A$ or $a_j(i) = A$ and $a_j(k) = H$.

We now proceed inductively. First we observe that the teams 1 and 3 can meet only in round 2 as after round 3 they have both the home games in even rounds and away games in odd rounds. In general, for any team i, $S(i, i+2) = i+1$ and hence there is a unique round in which the game between i and $i+2$ can be scheduled (team numbers are taken modulo $2n+1$ with the exception that 0 is replaced by $2n+1$). In particular, for $i = 1, 2, \ldots, 2n+1$ we have

to set $g(i, i+2) = i+1$. Now the teams 1 and 5 can play each other only in round 3: in rounds 1 and 5 one of them has a bye, in round 2 the team 1 plays the game against the team 3, and in round 4 the team 5 plays the game against the team 3. After round 5 their HAPs are equal. We can also check that for $i = 1, 2, \ldots, n$ we have $S(i, i+4) = \{i+1, i+2, i+3\}$ as the respective HAPs are equal before round i and after round $i+4$. But the game between i and $i+4$ cannot be played in round $i+1$, since there is the uniquely determined game $g(i, i+2)$. Or, in our notation, $i+1 = g(i, i+2)$. Also, this game cannot be scheduled for round $i+3$, as $i+3 = g(i+2, i+4)$. Therefore, we must have $g(i, i+4) = i+2$.

We continue inductively and suppose that for every $i = 1, 2, \ldots, 2n+1$ all values $g(i, i+2), g(i, i+4), \ldots, g(i, i+2s)$ have been uniquely determined. This indeed means that also the values $g(i, i-2), g(i, i-4), \ldots, g(i, i-2s)$ have been uniquely determined. We want to show that subsequently the game between i and $i+2s+2$ is also uniquely determined. We can assume here that $2s \leq 2n-1$ because of modularity. Then $S(i, i+2s+2) = \{i+1, i+2, \ldots, i+2s-1\}$. From our assumption it follows that $g(i, i+2) = i+1, g(i, i+4) = i+2, \ldots, g(i, i+2s) = i+s$. Also $g(i+2s, i+2s+2) = i+2s+1, g(i+2s-2, i+2s+2) = i+2s, \ldots, g(i+2, i+2s+2) = i+s+2$, and hence the game between i and $i+2s+2$ must be scheduled for round $i+s+1$. We notice here that because of modularity we get here also all games between teams i and $i+2t+1$, since $i+2t+1 \equiv i+2t-2n \pmod{2n+1}$. □

2.2 Even Number of Teams

One can now ask an obvious question: When it is possible to play an $RRT(2n+1, 2n+1)$ with no breaks, is it possible for an $RRT(2n, 2n)$ as well? The answer is affirmative. Although it may seem unnatural to construct a schedule that needs one more round than the necessary minimum, we can find a motivation in North-American collegiate competitions. The teams are divided into many conferences and it is required that conference games and non-conference games are distributed according to certain rules. Sometimes the non-conference games are scheduled before and after a block of conference games. However, some conferences have schedules where one or more non-conference games are scattered among conference games. Thus, the schedule of the conference games is usually non-compact.

The schedule is actually very simple and as in the case of an odd number of teams, it is also unique up to permutation of team numbers and reflection of the order of rounds. We first construct such a schedule and then prove the uniqueness.

Construction 4 Set $g(i, k) = g(k, i) = i + k - 1$. The addition is modulo $2n$ with the exception that 0 is replaced by $2n$. Obviously, $g(i, i) = 2i - 1$

Table 4. $RRT(8,8)$.

R 1	R 2	R 3	R 4	R 5	R 6	R 7	R 8
$8-2$	$5-6$	$1-3$	$6-7$	$2-4$	$7-8$	$3-5$	$8-1$
$7-3$	$4-7$	$8-4$	$5-8$	$1-5$	$6-1$	$2-6$	$7-2$
$6-4$	$3-8$	$7-5$	$4-1$	$8-6$	$5-2$	$1-7$	$6-3$
	$2-1$		$3-2$		$4-3$		$5-4$
1, 5 bye		2, 6 bye		3, 7 bye		4, 8 bye	

means that the team i has a bye in the round $2i-1$. So the teams with byes in the first rounds are 1 and $n+1$, and we choose as home teams for the first round the teams $2, 3, \ldots, n$. Notice that for $i = 1, 2, \ldots, n$ the teams i and $i+n$ have complementary home–away patterns with a bye in round $2i-1$. By setting $g'(i,k) = 2n+1-g(i,k)$ we get a tournament with byes in even rounds.

An example for eight teams is shown in Table 4.

Theorem 5 *For every $n \geq 2$ there exists an $RRT(2n, 2n)$ such that no $HAP(i)$ contains any sequence AA, HH, HBH, or ABA. Moreover, for each such n, the schedule is unique up to permutation of team numbers and reflection of the order of rounds.*

Proof. The existence was proved in Construction 4. Now we prove the uniqueness. Clearly, each team has exactly one bye, as there are $2n$ teams and $2n$ rounds. First we observe that there are at most two teams with a bye in each round. Obviously, the number of bye teams in each round must be even. Suppose there are at least four teams, i_1, i_2, i_3, and i_4, having a bye in round j. At least two teams of the quadruple i_1, i_2, i_3, i_4 play their first game either both away or both home. This is either in round 1 (if $j > 1$) or in round 2 (if $j = 1$). Suppose i_1 and i_2 play both an away game. Then their HAPs are equal and they can never play each other, because they play in each round either both a home game or both an away game. This contradicts our definition of a round robin tournament. We also observe that the two teams i, k that have a bye in a week j (recall that this is denoted by $a_j(i) = a_j(k) = B$) must have complementary schedules.

Now we show that there are at most two teams with a bye in any two consecutive rounds. Suppose it is not the case and there are teams i_1 and i_2 with a bye in a round j and k_1 and k_2 with a bye in the round $j+1$. Let $a_m(i_1) = A$ for some $m \neq j, j+1$. Then from the complementarity of HAPs of k_1 and k_2 it follows that $a_m(k_1) = A$ and $a_m(k_2) = H$ or vice versa. Suppose the former

Table 5. HAP for $RRT(2n, 2n)$.

	1	2	3	4	...	$2n-3$	$2n-2$	$2n-1$	$2n$
1	**B**	H	A	H	...	A	H	A	H
2	H	A	**B**	H	...	A	H	A	H
3	H	A	H	A	...	A	H	A	H
4	H	A	H	A	...	A	H	A	H
...					...				
...					...				
...					...				
$n-1$	H	A	H	A	...	**B**	H	A	H
n	H	A	H	A	...	H	A	**B**	H
$n+1$	**B**	A	H	A	...	H	A	H	A
$n+2$	A	H	**B**	A	...	H	A	H	A
...					...				
...					...				
...					...				
$2n-3$	A	H	A	H	...	H	A	H	A
$2n-2$	A	H	A	H	...	H	A	H	A
$2n-1$	A	H	A	H	...	**B**	A	H	A
$2n$	A	H	A	H	...	A	H	**B**	A

holds. Then the HAPs of the teams i_1 and k_1 are equal with the exception of rounds j and $j+1$. Therefore, they cannot play each other except possibly in round j or $j+1$. But $a_j(i_1) = B$ and $a_{j+1}(k_1) = B$ and hence they cannot play in rounds j or $j+1$ either. This is the desired contradiction.

Next we show that there are *exactly* two teams with a bye in any two consecutive rounds. In other words, we prove that the byes occur either in all odd rounds, or in all even rounds. We again proceed by contradiction. Suppose to the contrary that there are two consecutive rounds j and $j+1$ without byes. As there are no consecutive rounds *with* byes, it must happen that j is even and the byes occur precisely in rounds $1, 3, \ldots, j-1, j+2, \ldots, 2n$. But then there are teams i_1 and i_2 with $HAP(i_1) = BAHA \ldots HA$ and $HAP(i_2) = BHAH \ldots AH$ and also teams k_1 and k_2 with $HAP(k_1) = AHA \ldots HAB$ and $HAP(k_2) = HAH \ldots AHB$. Obviously, teams i_1 and k_2 can never play each other since their HAPs are equal except for weeks 1 and $2n$, when one of them has a bye. This contradiction shows that we can WLOG assume that byes occur in weeks $1, 3, \ldots 2n-1$.

Therefore, we define HAPs of respective teams as follows. For $i = 1, 2, \ldots, n$ we have $a_{2i-1}(i) = a_{2i-1}(n+i) = B$. For $i = 2, 3, \ldots, n$ we have $a_1(i) = H$ and $a_1(n+i) = A$. An example is shown in Table 5.

We again proceed by induction. First we observe that for any team i, $S(i, i+1) = \{i\}$ and hence there is a unique round in which the game between i and $i+1$ can be scheduled (team numbers are taken modulo $2n$ with the exception that 0 is replaced by $2n$). In particular, for $i = 1, 2, \ldots, n$ we have to set $g(i, i+1) = 2i$ and $g(n+i, n+i+1) = 2n - 2i$. We can also check that for $i = 1, 2, \ldots, n$ we have $S(i, i+2) = S(n+i, n+i+2) = \{2i, 2i+1, 2i+2\}$. But the game between i and $i+2$ cannot be played in round $2i$, since there is the uniquely determined game $g(i, i+1)$. Or, in our notation, $2i = g(i, i+1)$. Also, this game cannot be scheduled for round $2i+2$, as $2i+2 = 2(i+1) = g(i+1, i+2)$. The games between $n+1$ and $n+i+2$ can be argued similarly. Therefore, we must have $g(i, i+2) = g(n+i, n+i+2) = 2i+1$.

We can now continue inductively and suppose that for every $i = 1, 2, \ldots, 2n$ all values $g(i, i+1), g(i, i+2), \ldots, g(i, i+s)$ are uniquely determined. This indeed means that also the values $g(i, i-1), g(i, i-2), \ldots, g(i, i-s)$ are uniquely determined. We want to show that subsequently the game between i and $i+s+1$ is also uniquely determined. We can assume here that $s \leq n-1$ because of modularity. Then $S(i, i+s+1) = \{2i, 2i+1, \ldots, 2i+2s\}$. From our assumption it follows that $g(i, i+1) = 2i, g(i, i+2) = 2i+1, \ldots, g(i, i+s) = 2i+s-1$. Also $g(i+1, i+s+1) = 2i+s+1, g(i+2, i+s+1) = 2i+s+2, g(i+s, i+1) = 2i+2s$, and hence the game between i and $i+s+1$ must be scheduled for round $2i+s$. □

We observe that even if we consider a non-conference game to be scheduled in each conference bye slot, a schedule with the perfect HAP without breaks for more than two teams again cannot be found. The reason is the same as when we considered the compact schedule. Suppose there are more than two teams with a perfect HAP. Then two of them begin with a home game and no matter when they play their respective non-conference games, they again never play against each other.

In this paper we focused on schedules for 1-leg tournaments. Although there are competitions where 1-leg tournaments are widely used (e.g., chess tournaments, North-American collegiate football conferences, etc), 2-leg tournaments are much more common. It is natural to examine extensions of our schedules to 2-leg tournaments. The extension for $2n$ teams is easy and natural, because after swapping the home and away games in the second leg we get no breaks. For $2n+1$ teams, however, each team has a break between the first and second leg, that is, between the rounds $2n+1$ and $2n+2$. This can be avoided only by reversing the order of rounds in the second leg. This indicates that the new schedule for $2n$ teams, which we have constructed here may find its way to real life and we certainly hope it will.

Finally, we observe that if we number the teams and rounds $0, 1, \ldots, 2n$ or $0, 1, \ldots, 2n-1$, respectively, and disregard the home and away games, the game assignment function can be now defined in both cases as $g'(i, k) =$

$g'(k,i) = i + k$ which is corresponding to the additive group of order $2n + 1$ or $2n$, respectively.

Acknowledgments

Research for this paper was in part supported by the University of Minnesota Duluth Grant 177–1009 and by AGH University of Science and Technology Local Grant 11.420.04. The authors would like to thank the anonymous referees whose thoughtful comments and suggestions helped them to improve the quality of this paper.

References

de Werra, D. (1981) Scheduling in sports. In *Studies on Graphs and Discrete Programming*, North-Holland, Amsterdam, pp. 381–395.

Dinitz, J. and Froncek, D. (2000) Scheduling the XFL. *Congressus Numerantium*, **147**:5–15.

Finizio, N. J. (1993) Tournament designs balanced with respect to several bias categories. *Bulletin ICA*, **9**:69–95.

Griggs, T. and Rosa, A. (1996) A tour of European soccer schedules, or Testing the popularity of GK_{2n}. *Bulletin ICA*, **18**:65–68.

Henz, M. (1999) Constraint-based round robin tournament planning. In *Proceedings of the International Conference on Logic Programming*, MIT Press, Cambridge, MA, pp. 545–557.

Henz, M. (2001) Scheduling a major college basketball conference—revisited. *Operations Research*, **49**(1).

Henz, M., Müller, T. and Thiel, S. (2003) Global constraints for round robin tournament scheduling, *European Journal of Operations Research*, **153**:92–101.

Nemhauser, G. and Trick, M. A. (1998) Scheduling a major college basketball conference. *Operations Research*, **46**:1–8.

Schaerf, A. (1999) Scheduling sports tournaments using constraint logic programming. *Constraints*, **4**:43–65.

Schreuder, J. A. M. (1980) Construction timetables for sports competitions. In *Combinatorial Optimization II* (Conference Proceedings, University of East Anglia, 1979), *Mathematical Programming Study*, **13**:58–67.

Schreuder, J. A. M. (1992) Combinatorial aspects of construction of competition Dutch professional football leagues. *Discrete Applied Mathematics*, **35**:301–312.

Straley, T. H. (1983) Scheduling designs for a league tournament. *Ars Combinatoria*, **15**:193–200.

Trick, M. A. (2000) A Schedule-then-break approach to sports timetabling. In *Practice and Theory of Automated Timetabling III*, E. Burke and W. Erben (Eds.), Lecture Notes in Computer Science, Vol. 2079, Springer, Berlin, pp. 242–253.

UEFA (2004) www.uefa.com/uefa/members/

Transport Scheduling

RAIL CONTAINER SERVICE PLANNING: A CONSTRAINT-BASED APPROACH

Nakorn Indra-Payoong, Raymond S K Kwan, Les Proll
School of Computing, University of Leeds, Leeds, LS2 JT, UK
{ nakorn, rsk, lgp } @comp.leeds.ac.uk

Abstract This paper considers a container rail service planning problem, in which customer demands are known in advance. The existing rail freight optimisation models are complex and not demand responsive. This paper focuses on constructing profitable schedules, in which service supply matches customer demands and optimises on booking preferences whilst satisfying regulatory constraints. A constraint satisfaction approach is used, in which optimisation criteria and operational requirements are formulated as soft and hard constraints respectively. We present a constraint-based search algorithm capable of handling problems of realistic size. It employs a randomised strategy for the selection of constraints and variables to explore, and uses a predictive choice model to guide and intensify the search within more promising regions of the space. Experimental results, based on real data from the Royal State Railway of Thailand, have shown good computational performance of the approach and suggest significant benefits can be achieved for both the rail company and its customers.

Keywords: rail container service planning, local search, constraint-based approach.

1. INTRODUCTION

The transportation of rail freight is a complex domain, with several processes and levels of decision, where investments are capital-intensive and usually require long-term strategic plans. In addition, the transportation of rail freight has to adapt to rapidly changing political, social, and economic environments. In general, the rail freight planning involves four main processes: path formulation, fleet assignment, schedule production, and fleet repositioning. This paper addresses an issue in schedule production, constructing profitable schedules for the container rail service, using a constraint-based approach.

1.1 Container Rail Service Planning Problem

Container rail service differs from conventional freight rail in several important aspects. Because of the high costs of container handling equipment,

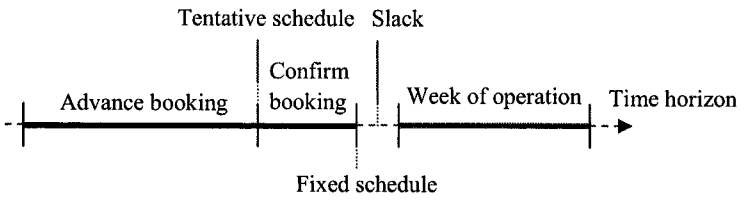


Figure 1. A short-term advance booking scheme.

container rail networks have relatively few, and widely spaced, terminals. Networks with a small number of terminals are common and the network flows are relatively simple. A typical container makes few or no stops and may be transferred between trains only up to a few times on its journey. In addition, small lot sizes of shipment, frequent shipment, and demand for flexible service are important characteristics of rail container transportation.

This paper considers the container rail service from a container port to an inland container depot (ICD). Once containers arrive at the port, there is a need to move them to their final customers, which can be done by rail or truck via ICD, or by truck direct to the final destinations.

A rail carrier's profitability is influenced by the railway's ability to construct schedules for which service supply matches customer demand. The need for flexible schedules is obvious because the take-up of some services in a fixed schedule may be low and not profitable. In order to create a profitable timetable, a container rail carrier needs to engage in a decision-making process with multiple criteria and numerous constraints, which is very challenging.

This paper assumes an advance booking scheme as illustrated in Figure 1. It also assumes that all containers are homogeneous in terms of their physical dimensions, and they will be loaded on trains ready for any scheduled departure times.

Customers are requested to state a preferred departure timeslot or an earliest departure time in advance. A number of alternative departure timeslots for each shipment may be specified, which might be judged from experience or estimated by the customer's delay time functions. These alternatives not only help a rail carrier consolidate customer demands to a particular train service with minimum total costs, but also provide flexible departure times for the customer's transport planning strategy.

1.2 Literature Review

There are two principal areas of work relating to the application domain and the solution approach.

Application domain. There is an increasing interest in flexible rail freight schedules in the literature, which may be distinguished into two types according to how the overall demand is met. Huntley *et al.* (1995), Gorman (1998) and Arshad *et al.* (2000) aggregate customers with minimum operating costs through flexible scheduling. They do not propose to meet individual demands. Newman and Yano (2000), Yano and Newman (2001) and Kraft (2002) share the same spirit of our study by being responsive to individual demands.

The models proposed by Newman and Yano (2000), Yano and Newman (2001) and Kraft (2002) satisfy the operational constraints fully for each customer. In contrast, our framework models customer satisfaction, computed from preferred and alternative departure times, which is then maximised as one of the business criteria. Hence, some customers might not be given their most preferred departure times. This framework is more natural for supporting decision-makers, in which a rail carrier can measure how well their customers are satisfied and the implications of satisfying these customers in terms of cost.

Solution approach. As the size of the problem that needs to be routinely solved in the rail freight industry is large, local search methods, such as simulated annealing, tabu search, genetic algorithms, etc, have been employed to produce near-optimal solutions. For instance, Huntley *et al.* (1995) applied simulated annealing to the problem of rail freight routing and scheduling, Marin and Salmeron (1996) evaluated a descent method, simulated annealing, and tabu search for solving large size rail freight networks, Gorman (1998) used a tabu-enhanced genetic algorithm for solving the freight railroad operating plan problem, and Arshad *et al.* (2000) combined constraint programming with a genetic algorithm to solve the multi-modal transport chain scheduling problem. A recent survey of optimisation models for train routing and scheduling is given by Cordeau *et al.* (1998). However, these approaches are complex and highly domain specific; thus they lose flexibility and the sustainability to solve rail container optimisation models in which the rail business strategy keeps changing.

The concept of domain-independent algorithms is always attractive and may be appropriate for our problem. There are many algorithms in this class; for example, Connolly (1992) introduced general purpose simulated annealing (GPSIMAN), Abramson and Randall (1999) extended GPSIMAN to solve integer programs (INTSA), and Nonobe and Ibaraki (1998) proposed a tabu search approach as a general solver for a constraint satisfaction problem.

In contrast, our approach is inspired by SAT local search for the satisfiability (SAT) problem (Gomes *et al.*, 1998; Selman *et al.*, 1992, 1994). SAT is a problem of deciding whether a given Boolean formula is satisfiable. When the problem is not solvable in polynomial time by exact algorithms, SAT local search might be employed. An attractive framework of SAT local search is that

the structure of the local move is simple. GSAT and WalkSAT are well-known local search techniques for SAT problems (Selman *et al.*, 1992, 1994). SATz-Rand, introduced by Gomes *et al.* (1998) is a recent solver for SAT problems. Walser (1999) extended WalkSAT to WSAT(OIP) for solving integer programming problems.

However, our problem encoded into a Boolean formula or 0–1 integer constraints would be large and the solution structure may be difficult to maintain by simple local move with a randomised strategy, as performed by SAT local search and other domain-independent algorithms. One way to enhance the algorithm whilst maintaining a simple structure of local move is to build in learning capabilities in the algorithm. We propose a predictive choice model that learns from the search history in order to fix locally some variables, and enforce the consistency between different sets of variables. The model addresses the case in which all decision variables are binary. Our predictive learning algorithm has some similarities to other algorithms based on probabilistic models (Horvitz *et al.*, 2001; Larraaga and Lozano, 2002; Marin and Salmeron, 1996; Resende and Ribero, 2001). All these algorithms attempt to draw inferences specific to the problem and therefore can be regarded as processes for learning domain knowledge implicitly. However, the formulation and application of the models are different. Our predictive model is based on a discrete choice theory of human behaviour for choosing a particular value for variables in a probabilistic way, whilst the others are based on different theories and use the probabilistic models to limit the search space at different points of process.

The paper is organised as follows. We first define the hard and soft constraints and model the problem as a constraint satisfaction problem. The solution algorithm will be described next. Experimental results, based on real data from the Royal State Railway of Thailand will be presented. Finally, conclusions are discussed.

2. CONSTRAINT-BASED MODELLING

Real-world problems tend to have a large number of constraints, which may be hard or soft. Hard constraints require that any solutions will never violate the constraints. Soft constraints are more flexible, constraint violation is tolerated but attracts a penalty. Naturally, a real-world problem can be thought of as a constraint satisfaction problem (CSP).

There are two critical advantages of using constraint-based modelling. Firstly, it is a clean separation between problem modelling and solution technique. If new problem conditions are introduced, we only need to model such conditions as constraints. Secondly, problem-specific knowledge can influence the search naturally. This is done by applying problem-specific weights,

reflecting their relative importance, directly to constraints in order to enhance a solution algorithm within a CSP framework.

We model the demand-responsive container rail scheduling problem as a CSP and introduce a constraint-based search for solving this class of CSP. We consider problems in which the day is divided into hourly slots for weekly booking and scheduling. The following notation will be used.

Subscripts:

- t schedulable timeslot (departure time), $t = 1, 2, 3, \ldots, T$.
- j customer, $j = 1, 2, 3, \ldots, M$.

Sets:

- S_j set of possible departure timeslots for customer j.
- C_t set of potential customers for departure timeslot t.
- R set of service restrictions for departure timeslots.

Decision variables:

- x_t 1, if a train departs in timeslot t, 0 otherwise.
- y_{tj} 1, if customer j is served by the train departing in timeslot t, 0 otherwise.

Parameters:

- w_{tj} customer j satisfaction in departure timeslot t.
- r_t train congestion cost in departure timeslot t.
- g_t staff cost in departure timeslot t.
- P_2 capacity of a train (number of containers).
- N_j demand of customer j (number of containers).

Our problem may be thought of as analogous to a capacitated facility (warehouse) location problem (CFLP) (Aardal, 1998; Beasley, 1988), with the container shippers being the customers, and the train departure timeslots being analogous to the possible warehouse locations. The CFLP class of problems also include location and distribution planning, capacitated lot sizing in production planning, and communication network design (Boffey, 1989; Kochmann and McCallum, 1981), etc. However, in contrast to the CFLP, our problem is recurrent with complex interaction between the possible warehouse locations. We handle non-uniform demands that arrive at the container port dynamically with distinct target times to their final destinations. In addition, we include in our model a probabilistic decrease in customer satisfaction as the schedulable timeslots deviate from the customer target time.

In a CSP-model, optimisation criteria and operational requirements are represented as soft and hard constraints respectively. The criteria are handled by transforming them into soft constraints. This is achieved by expressing each

criterion as an inequality against a tight bound on its optimal value. As a result, such soft constraints are rarely satisfied.

A feasible solution for a CSP representation of the problem is an assignment to all constrained variables in the model that satisfies all hard constraints, whereas an optimal solution is a feasible solution with the minimum total soft constraint violation (Henz *et al.*, 2000; Lau *et al.*, 2001; Walser, 1999). For a constraint satisfaction problem, the violation ν_i of constraint i is defined as follows:

$$\sum_j a_{ij}x_j \leq \ b_i \Rightarrow \nu_i = \max\left(0, \sum_j a_{ij}x_j - b_i\right) \tag{1}$$

where a_{ij} are coefficients, b_i is a tight bound and x_j are constrained variables. Note that violations for other types of linear and non-linear constraints can be defined in an analogous way.

When all constrained variables are assigned a value, the violation of the hard and soft constraints can be tested and quantified for evaluating local moves.

2.1 Soft Constraints

The number of trains. The aim is to minimise the number of trains on a weekly basis, which is defined as

$$\sum_{t=1}^{T} x_t \leq \theta \tag{2}$$

where θ is a lower bound on the number of trains, e.g. $\left\lceil \left(\sum_j N_j\right) / P_2 \right\rceil$.

Customer satisfaction. This constraint aims to maximise the total customer satisfaction. The satisfaction is assigned values from a customer satisfaction function (e.g., Figure 2). Each customer holds the highest satisfaction at a preferred booking timeslot, the satisfaction then decreases probabilistically to the lowest satisfaction at the last alternative booking timeslot, i.e. later than preferred booking timeslots would cause a decrease in the future demand, and the rail carrier is expected to take a loss in future revenue. For the evaluation of a schedule, the probability of customer satisfaction is then multiplied by demand N_j. The customer satisfaction constraint can be expressed as

$$\sum_{t=1}^{T}\left(\sum_{j\in C_t} w_{tj}y_{tj}N_j\right) \geq \Omega \tag{3}$$

where Ω is an upper bound on customer satisfaction, i.e. $\sum_j W_j N_j$, with W_j the maximum satisfaction on the preferred booking timeslot for customer j.

Timeslot operating costs. This constraint aims to minimise the operating costs. A rail carrier is likely to incur additional costs in operating a demand responsive schedule, in which departure times may vary from week to week. This may include train congestion costs and staff costs. The train congestion cost reflects an incremental delay resulting from interference between trains in a traffic stream. The rail carrier calculates the marginal delay caused by an additional train entering a particular set of departure timeslots, taking into account the speed-flow relationship of each track segment. The over-time costs for crew and ground staff would also be paid when evening and night trains are requested. The constraint is defined as

$$\sum_{t=1}^{T} (r_t + g_t)\, x_t \leq (\lambda + \delta) \tag{4}$$

where $(\lambda + \delta)$ is a lower bound on the timeslot operating costs, $\lambda = \sum_{t \in T_a} r_t$, T_a is the set of θ least train congestion costs, $\delta = \sum_{t \in T_b} g_t$, T_b is the set of θ least staff costs, θ is a lower bound on the number of trains.

2.2 Hard Constraints

Train capacity. This constraint ensures the demand must not exceed the capacity of a train, which is defined as

$$x_t \left(\sum_{j=1}^{M} y_{tj} N_j \right) \leq P_2 \quad \forall t \tag{5}$$

Coverage constraint. It is a reasonable assumption that in practice customers do not want their shipment to be split in multiple trains, this constraint ensures that a customer can only be served by one train. The constraint is given as

$$\sum_{t \in S_j} y_{tj} = 1 \quad \forall j \tag{6}$$

Timeslot consistency. This constraint ensures that if timeslot t is selected for customer j, a train does depart at that timeslot. On the other hand, if departure time t is not selected for customer j, a train may or may not run at that time. The constraint is defined as

$$x_t \geq y_{tj} \quad \forall j \in C_t \tag{7}$$

Service restriction. This is a set of banned departure times. The restrictions may be pre-specified so that a railway planner schedules trains to

achieve a desirable headway or to avoid congestion at the container terminal. The constraint is defined as

$$x_t = 0 \quad \forall t \in R \tag{8}$$

2.3 Implied Constraints

The soft and hard constraints completely reflect the requisite relationships between all the variables in the model, i.e. the operational requirements and business criteria. Implied constraints, derivable from the above constraints, may be added to the model. While implied constraints do not affect the set of feasible solutions to the model, they may have computational advantage in search-based methods as they reduce the size of the search space (Proll and Smith, 1998; Smith *et al.*, 2000).

Timeslot covering. A covering constraint can be thought of as a set covering problem in which the constraint is satisfied if there is at least one departure timeslot x_t serving customer j. This constraint favours a combination of selected departure timeslots that covers all customers. The covering constraint is defined as

$$\sum_{t \in S_j} x_t \geq 1 \quad \forall t \tag{9}$$

2.4 Customer Satisfaction

A rail carrier could increase the quality of service by tailoring a service that satisfies customers. The rail schedule may be just one of the factors including cost, travel time, reliability, safety, and so forth. As customers have different demands, it is hard to find a single definition of what a good quality of service is. For example, some customers are willing to tolerate a delayed service in return for sufficiently low total shipping costs.

In this paper, we only investigate customer satisfaction with respect to the rail schedule. To acquire the customer satisfaction data, face-to-face interviews were carried out. This survey includes 184 customers currently using both rail and trucking services or using only rail but with the potential to ship by truck in the future. To quantify customer satisfaction, customer satisfaction functions were developed. Total shipping costs associated with movement by different modes are calculated as a percentage of commodity market price or value of containerised cargo, expressed in price per ton. Average shipping costs of the cargo from survey data and the market price (Ministry of Commerce, 2002) are summarised in Table 1.

We assume that all customers know a full set of shipping costs, and can justify modal preferences on the basis of accurately measured and understood costs. The freight rate may be adjusted by the relative costs that a customer

Table 1. Modal cost for a transport mode.

Cargo types/cost	Cost /unit price Truck	Rail	Market price	Modal cost (%) Truck, C_T	Rail, C_R	ΔC
Freight rate						
Type I	2.21	1.55	25.00	8.84	6.20	2.64
Type II	6.71	2.96	68.00	9.87	4.35	5.52
Type III	10.45	7.56	87.20	11.98	8.67	3.31
Type IV	0.95	0.21	13.00	7.30	1.62	5.68
Terminal handling charge						
Type I	0.28	0.51	25.00	1.12	2.04	−0.92
Type II	0.57	1.04	68.00	0.84	1.53	−0.69
Type III	1.18	2.06	87.20	1.35	2.36	−1.01
Type IV	0.03	0.08	13.00	0.23	0.61	−0.38
Terminal storage charges						
(within free time storage)	0	0		0	0	0
Overhead cost						
(within free time storage)	0	0		0	0	0
Total shipping costs						
Type I	2.49	2.06	25.00	9.96	8.24	1.72
Type II	7.28	4.00	68.00	10.70	5.88	4.82
Type III	11.63	9.62	87.20	13.34	11.03	2.31
Type IV	0.98	0.29	13.00	7.54	2.23	5.31

may be willing to pay to receive superior service. For example, some customers may have higher satisfaction using a trucking service even if the explicit freight rate is higher; speed and reliability of the service may be particularly important if the containerised cargo has a short shelf life.

To determine customer satisfaction between modes, modal cost percentages are then applied to an assumed normal distribution (Indra-Payoong *et al.*, 1998) and the difference between modal cost percentages, i.e. $\Delta C = C_T - C_R$. The customer satisfaction derived from the cumulative probability density function is illustrated in Figure 2.

Once the satisfaction function has been developed, a customer satisfaction score can be obtained from the modal satisfaction probability. This probability could also be used to predict the market share between transport modes and to test the modal sensitivity when the rail schedule is changed.

The customer satisfaction is a probability of choosing rail service; hence satisfaction ranges from 0 to 1. Note that all customers currently using container rail service may already hold a certain level of satisfaction regardless of taking the quality of rail schedule into account. Once the rail carrier has been chosen as transport mode and later the schedule is delayed, customers incur additional

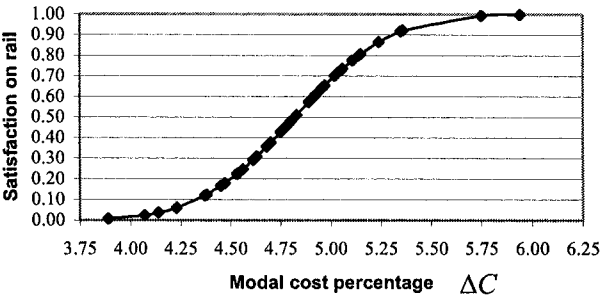


Figure 2. Customer satisfaction function of cargo type II.

total shipping costs, i.e. terminal storage and overhead costs involved at the seaport. This would result in a decrease in customer satisfaction.

2.5 Generalised Cost Function

For the evaluation of a schedule, a cost function taking into account the number of trains can be expressed in terms of operating costs; but it is hard to express customer satisfaction using a monetary unit. We shall express the customer satisfaction on a rail scheduling service in terms of shipping costs related to the delay time. We introduce the term "virtual revenue loss" as a unit cost. This term is derived from the difference in probability of choosing the rail service between the preferred timeslot and the alternatives. The probability is then multiplied by a demand and freight rate per demand unit. Therefore, a generalised cost function, GC, is the sum of the operating costs and the virtual loss of revenue:

$$GC = (\theta + s_1)FC + s_2FR + (\lambda + \delta + s_3) \tag{10}$$

where FC is a fixed cost of running a train, FR is a freight rate per demand unit (ton-container), s_1, s_2 and s_3 are soft constraint violations for the number of trains, customer satisfaction, and timeslot operating costs constraints respectively.

3. SOLUTION ALGORITHM

We propose a constraint-based search algorithm (CBS) for solving the constraint satisfaction problem. The algorithm consists of two parts: CBS based on a randomised strategy and a predictive choice learning method, which guides and intensifies the search.

3.1 Constraint-Based Search Algorithm

Points in the search space correspond to complete assignment of 0 or 1 to all decision variables. The search space is explored by a sequence of sample randomised moves which are influenced by the violated hard constraints at the current point.

The CBS starts with an initial random assignment, in which some hard constraints in the model can be violated. In the iteration loop, the algorithm randomly selects a violated constraint: e.g., the assigned train timeslot for which the demands exceed train capacity. Although different constraint selection rules have been studied for SAT local search (McAllester *et al.*, 1997; Parkes and Walser, 1996; Walser, 1999), for instance choosing the violated constraint with maximum or minimum violation, none have been shown to improve over random selection.

Having selected a violated constraint, the algorithm randomly selects one variable in that constraint and another variable, either from the violated constraint or from the search space. Then, two flip trials are performed, i.e. changing the current value of the variable to its complementary binary value. Suppose that V_i takes the value v_i at the start of the iteration, so that $A = (v_1, v_2, \ldots, v_m|h)$, where m is the total number of variables and h is the total violation of all hard constraints. Suppose further that V_1,V_2 are chosen and that their flipped values are $\bar{v}_1$, $\bar{v}_2$ respectively. We then look at the assignments $A_1 = (\bar{v}_1, v_2, \ldots, v_m|h_1)$, $A_2 = (v_1, \bar{v}_2, \ldots, v_m|h_2)$ and select the alternative with the smaller total hard violation. Whenever all hard constraints are satisfied, the algorithm stores the soft violation penalties as feasible objective values, together with the associated variable values. The algorithm continues until the stopping criterion is met, i.e. a feasible solution is found or if no improvement has been achieved within a specified number of iterations. The procedure of CBS is outlined in Figure 3.

The procedure can be readily modified to give a set of feasible solutions and to make more use of the soft constraints, which in the procedure of Figure 3 are largely ignored. We do not do so here but do in the context of an enhanced procedure incorporating the predictive choice model in the following section.

3.2 Predictive Choice Model

The first development of choice models was in the area of psychology (see Marder, 1997). The development of these models arose from the need to explain the inconsistencies of human choice behaviour, in particular consumer choice in marketing research. If it were possible to specify the causes of these inconsistencies, a deterministic choice model could be easily developed.

These causes, however, are usually unknown or known but very hard to measure. In general, these inconsistencies are taken into account as random

```
proc CBS
  input soft and hard constraints
  A    := initial random assignment
  while not stopping criterion do
    C := select-violated-hard-constraint (A)
    P := select-two-variables (C, A)
    A1,   A2    := flip(A, P)
    if (h1 < h2) then (A ← A1)
    else (A ← A2)
    if h = 0 then A is feasible, record solution A
    end if
  end while
  output a feasible solution found
end proc
```

Figure 3. The constraint-based search procedure.

behaviour. Therefore, the choice behaviour could only be modelled in a probabilistic way because of an inability to understand fully and to measure all the relevant factors that affect the choice decision.

Deciding on a choice of value for a variable in a CSP is not obviously similar to the consumer choice decision. However, we could set up the search algorithm to behave like the consumer behaviour in choice selection. That is, we consider the behavioural inconsistencies of the algorithm in choosing a good value for a variable.

For general combinatorial problems, a particular variable may take several different values across the set of feasible solutions. Thus it may never be possible to predict a consistently good value for the variable during the search. However, when the problem is severely constrained and has few feasible solutions, it may well be that some variables take a more consistent value in all the feasible solutions during the search. The predictive choice model is intended to discover such values and to use them to steer the search. Note that for the container rail scheduling model in Section 2, the problem becomes more severely constrained as the value of minimum train loading increases (Section 3.3).

Violation history. Once a variable has been selected, the algorithm has to choose a value for it. The concept is to choose a good value for a variable: e.g. the one that is likely to lead to a smaller total hard constraint violation in a complete assignment. In our constraint-based search algorithm, two variables are considered at each flip trial. The first variable is randomly chosen

Table 2. Violation history.

	Variable of interest x_1				Compared variable x_j					
Flip	Current		Flipped			Current		Flipped		
trial	Val	h_1	Val	h_1'	j	Val	h_2	Val	h_2'	x_1^*
1	1	26	0	22	15	1	26	0	36	0
2	1	20	0	12	9	0	20	1	6	1
3	1	15	0	14	30	0	15	1	10	1
N	0	46	1	53	8	0	46	1	31	0

from those appearing in a violated constraint and considered as the variable of interest; the second variable is randomly selected, either from that violated constraint or from the search space, and is to provide a basis for comparison with the variable of interest.

Clearly, the interdependency of the variables implies that the effect of the variable value chosen for any particular variable in isolation is uncertain. Flipping the first variable might result in a reduction in total hard constraint violation. However, it might be that flipping the second variable would result in even more reduction in the violation. In this case, the flipped value of the first variable is not accepted.

In Table 2, the variable of interest is x_1 and the compared variable is x_j; the two variables are trial flipped in their values; the violations associated with their possible values are recorded and compared. In this table, h is the total hard violation, $x_1^*$is the value of x_1 chosen in the flip trial. Note that only h_1, h_1', and x_1^* are recorded for the violation history of x_1.

In flip trial 1 the selected variables are x_1 (current value 1) and, separately, x_{15} (current value 1). The current assignment has violation = 26. Flipping x_1, with x_{15} fixed at 1, gives violation = 22; flipping x_{15}, with x_1 fixed at 1, gives violation = 36. Hence in this trial the algorithm records $x_1 = 0$ as the better value. At some later iteration the algorithm chooses to flip x_1 again, this time (flip trial 2) with compared variable x_9. Flipping x_1, with x_9 fixed at 0, gives violation = 12; flipping x_9, with x_1 fixed at 1, gives violation = 6. Although flipping x_1 to 0 gives a better violation than the current assignment, in this flip trial the algorithm records $x_1 = 1$ as the better value as there is an assignment with $x_1 = 1$ which gives an even better violation. If we view the results of these flip trials as a random sample of the set of all assignments, we can build up a predictive model to capture the behavioural inconsistency in the choice selection and to predict what would be a "good" value for x_1.

Utility concept. The predictive choice model is based on the random utility concept (Anderson *et al.*, 1992; Ben-Akiva and Lerman, 1985). Choosing a good value for a variable in each flip trial is considered as a non-deterministic task of the search algorithm. The algorithm is designed to select a choice of value for a variable that has a maximum utility.

However, the utility is not known by the algorithm with certainty and is therefore treated as a sum of deterministic and random utilities. The utility is defined as follows:

$$U_0 = V_0 + \varepsilon_0 \tag{11}$$

where U_0 is an overall utility for the algorithm choosing value 0, V_0 is a deterministic utility for the algorithm choosing value 0, ε_0 represents inconsistencies (uncertainties) in the choice selection, measurement errors and unobserved choice decision factors, and is a random utility for the algorithm choosing value 0.

For each flip trial, the algorithm selects value 0, when flipping a variable to 0 is preferred to 1. This can be written as follows:

$$0 \succ 1 \Rightarrow U_0 > U_1 \Rightarrow (V_0 + \varepsilon_0) > (V_1 + \varepsilon_1) \tag{12}$$

The random utilities ε_0 and ε_1 may cause uncertainty in the choice selection, i.e. U_0 might be greater than U_1 or U_1 might be greater than U_0, even if the deterministic utility satisfies $V_0 > V_1$. From this point, the probability for the algorithm choosing value 0 is equal to the probability that the utility of choosing value 0, U_0, is greater than the utility of choosing value 1, U_1. This can be written as follows:

$$P_0 = \text{Prob}[U_0 > U_1] \tag{13}$$

where P_0 is the probability for the algorithm choosing value 0.

Thus,

$$P_0 = \text{Prob}\left[(V_0 - V_1) > (\varepsilon_1 - \varepsilon_0)\right] \tag{14}$$

To derive a predictive choice model, we require an assumption about the joint probability distribution of the random utilities ε_1 and ε_0.

Joint probability distribution. To derive the joint probability distribution of the random utilities ε_1 and ε_0, the difference between the random utilities, i.e. $\varepsilon' = \varepsilon_1 - \varepsilon_0$, is used. However, ε' is unknown by the algorithm. We use the difference between deterministic utilities, i.e. $V' = V_1 - V_0$, to investigate the probability distribution of ε' because the deterministic and random utilities are the components of the overall utility U.

We can now look for an appropriate functional form for the distribution of V'. From the central limit theorem (Trotter, 1959), whenever a random sample of size n ($n > 30$) is taken from any distribution with mean μ and variance

σ^2, then the sample would be approximately normally distributed. We perform the Shapiro–Wilk test and Kolmogorov–Smirnov test (Patrick, 1982; Shapiro and Wilk, 1965) to find out whether the non-deterministic component appears to follow any specific distribution. From our experiments and the central limit theorem, we are first encouraged to assume normality of the distribution.

Although the normal distribution seems reasonable based on the central limit theorem, it has a problem with not having a closed probability function form, i.e. the PDF is expressed in terms of an integral; thereby it is computationally intractable. The logistic function is therefore chosen instead because its distribution is an approximation of the normal law (Kallenberg, 1997). Under the assumption that V' is logistically distributed, applying a standard logistic distribution function and probability theory, a specific probabilistic choice model, the *logit model* (Anderson *et al.*, 1992; Ben-Akiva and Lerman, 1985), can be obtained as follows:

$$P_0 = \frac{e^{V_0}}{e^{V_0} + e^{V_1}} \tag{15}$$

where P_0 is the probability for the algorithm choosing value 0.

Violation function. For any flip trial, the deterministic utility V may be characterised by many factors. In this research, the utility is determined by the total hard constraint violation h. This is because it can easily be measured by the algorithm and gives a reasonable hypothesis to the choice selection. In other words, we would like to use a function of deterministic utility for which it is computationally easy to estimate the unknown parameters.

We define a function that is linear in its parameters. A choice specific parameter is introduced so that one alternative is preferred to the other when the total hard violation is not given, i.e. the choice decision may be explained by other factors. The deterministic utility functions for V_0 and V_1 are defined as

$$V_0 = \beta_1 + \beta_2 h_0 \tag{16}$$

$$V_1 = \beta_2 h_1 \tag{17}$$

where β_1 is the choice specific parameter, β_2 is the violation parameter, and h_0 and h_1 are the total hard violations when a variable is assigned a value to 0 and 1 respectively.

We could now use the predictive choice model to predict a value for a particular variable from an individual flip trial. However, the predictions for an individual flip trial may not reliably help the algorithm make a decision on what a good value for a variable would be. Instead, we use an aggregate quantity, i.e. a prediction for the value choice based on a set of trials. We use the arithmetic mean of the total hard violation to represent the aggregate of N flip

trials, which can be written as

$$\bar{h}_0 = \sum_{n=1}^{N} \frac{h_{0,n}}{N} \quad \text{and} \quad \bar{h}_1 = \sum_{n=1}^{N} \frac{h_{1,n}}{N} \tag{18}$$

where $\bar{h}_0$ and $\bar{h}_1$ are the average total hard violations when a variable is assigned a value 0 and 1 respectively.

Logit method. When an occurrence of any choice value x^* is not obviously dominating (Table 2), the logit method is called. As a statistical way to estimate the utility's parameter values requires a significant computational effort, we introduce a simplified estimation, in which logit method only accounts for the constraint violation. We set the choice-specific parameter β_1 to any small value, e.g. $\beta_1 = 0.05$, so that the utility of one alternative is preferred to the other. This is because an equal utility lies outside the assumption of the choice theory. Then, the relative difference between $\bar{h}_0$ and $\bar{h}_1$, $\Delta\bar{h}$, is used in order to characterise the value choice selection. $\Delta\bar{h}$ is defined as follows:

$$\Delta\bar{h} = \left| \frac{\bar{h}_0 - \bar{h}_1}{\bar{h}_0 + \bar{h}_1} \right| \tag{19}$$

where $\bar{h}_0$ and $\bar{h}_1$ are the average total hard violations when a variable is trial flipped or assigned a value 0 and 1 respectively.

From (19), when the value of $\Delta\bar{h}$ is large, the probabilities of two alternatives (value 0 and 1) would be significantly different, and when $\Delta\bar{h}$ = 0, the probabilities of the two alternatives would tend to be equal. $\Delta\bar{h}$ is shown in a proportional scale so that the formulation could be generalised for a combinatorial problem in which the total hard violation and a number of flip trials can be varied. Then, we use a simplified estimation of β_2 as follows:

$$\beta_2 = -\Delta\bar{h} \tag{20}$$

where β_2 is the violation parameter.

Proportional method. This method is also based on a probabilistic mechanism in the sense that the algorithm may select the current value of the variable even though flipping that variable to the other value gives a lower violation.

The proportional method is more straightforward than the logit method. The choice selection is only affected by the number of occurrences of choice values in x^*, i.e. the constraint violation is not explicitly considered. This method is developed and used to enhance the choice prediction by the simplified estimation of the utility's parameters when $\bar{h}_0$ and $\bar{h}_1$ are close. In this case

the logit method may not perform well. In addition, this method requires less computation than the logit method. The proportional method is defined as

$$P_0 = \frac{x_0^*}{N} \tag{21}$$

where P_0 is the probability for the algorithm choosing value 0, x_0^* is the number of occurrences of value 0 in x^*, N is the number of flip trials.

An outcome of the predictive choice model is the probability of choosing a particular value for a variable. The timeslot and customer's booking variables (x_t and y_{tj}) are chosen for flip trials, but propagation of their values for consistency may not be carried out fully all the time. At the beginning, constraint propagation is only carried out within each of the sets x_t and y_{tj}, but not across the two sets of variables, in order to promote wider exploration of the search space.

After a specified number of iterations, the trial violation history is analysed. Some variables may have high probability of a particular value given by the predictive choice model. These variables will be fixed at their predicted value for a number of iterations determined by the magnitude of the associated probability. At this point, consistency between timeslots and customer's bookings variables is enforced, leading to intensified exploration of the neighbouring search space. When the fixing iteration limit, F, is reached, the variable is freed and its violation history is refreshed.

3.3 Minimum Train Loading

The constraint-based search assigns a fixed number of trains according to the number of trains expected, which is derived from the minimum train loading. In other words, a fixed number of timeslots used is maintained during the search process, which can be written as

$$\sum_{t=1}^{T} x_t = T_{\text{exp}} \tag{22}$$

where T_{exp} is the number of trains expected.

Setting a minimum train loading ensures satisfactory revenue for a rail carrier and spreads out the capacity utilisation on train services. The carrier may want to set the minimum train loading as high as possible, ideally equal to the capacity of a train. Note that the minimum train loading is directly related to T_{exp}, which is defined as

$$T_{\text{exp}} = \left\lceil \sum_j N_j / P_1 \right\rceil \tag{23}$$

where N_j is the demand of customer j, and P_1 is the minimum train loading.

Apart from ensuring satisfactory revenue, minimum train loading is a key factor in the performance of the search algorithm. The higher the minimum train loading, the more constrained the problem is and hence the number of feasible solutions decreases. Using a high minimum train loading allows the algorithm to focus on satisfying the hard constraints more than the soft constraints. In addition, it increases the usefulness of the value choice prediction mechanism, i.e. the variables in the container scheduling model would take more consistent values in all the feasible solutions during the search.

However, it would be very hard to prove whether there exists a feasible solution to the problem constrained by a high minimum train loading. If we could prove the existence of a feasible solution for the highest possible minimum train loading, it would imply that the solution is approximately optimal. A good setting of the minimum train loading helps limit the size of the search space. Although a few techniques for proving the existence of feasibility have been proposed (Hansen, 1992; Kearfott, 1998), implementations of these techniques for practical problems have not yet been achieved. In this research, the minimum train loading is derived from some heuristic rules. We estimate the minimum train loading by defining a risk parameter R. For example, $R = 20\%$ means that the estimated chance of the problem having no feasible solution is 20%. An initial value of P_1 is defined as follows:

$$P_1 = \frac{\sum_j N_j}{\lceil M/T \rceil} \tag{24}$$

where $T = \left\lfloor \frac{P_2}{\mu_R} \right\rfloor$, $\mu_R = \frac{(\mu_g + \sigma_g) \times (100 - R)}{100}$, P_2 is a capacity of a train, N_j is the demand of customer j, M is the total number of customers, and μ_g and σ_g are the mean and standard deviation of the total average demand.

Whenever all hard constraints are satisfied (a feasible train schedule is obtained), the minimum train loading is implicitly increased by removing one train from the current state of the feasible solution, i.e. $T_{\text{exp}} = T_{\text{exp}} - 1$, and CBS attempts to find a new feasible schedule.

4. HIERARCHICAL CONSTRAINT SCHEME

In SAT local search and its variants, the number of violated constraints (unsatisfied clauses) is used to evaluate local moves without accounting for how severely individual constraints are violated. In CBS, a quantified measure of the constraint violation to evaluate local moves is used. In this case, the violated constraints may be assigned different degrees of constraint violation. This leads to a framework to improve the performance of the solving algorithm. The constraints can be weighted in the formulation of train measures of

violation in order to allow the search to give hierarchical priority to satisfying some subsets of the constraints.

For the container rail service planning, soft and hard constraints in the model are treated separately. When all hard constraints are satisfied, the soft constraint violations are calculated and used as a measure of the quality of the solution. Nevertheless, whilst the hard constraint have not yet been fully satisfied, our scheme incorporates an artificial constraint, and its weighted violation measure is designed to exert some influence over the search process based on an estimation of the soft constraint violation (Section 4.2).

4.1 Feasibility Weights

The principal goal of the CBS is to find feasible solution to the problem, i.e. points at which the total violation of the hard constraints is zero. For the container rail service planning model, all sets of hard constraints use weighted measures of violation according to some heuristic rules.

From (5), any number of containers in a potential timeslot exceeding a train capacity is penalised with the same violation h_m. An attempt to use different measures of the violation to different number of exceeded containers on an assigned train makes little sense because one can never guarantee whether the lower number of exceeded containers is more likely to lead to feasible schedules. The violation penalty for a set of capacity constraints is defined as

$$x_t \left(\sum_{j=1}^{M} y_{tj} N_j \right) \begin{cases} \leq P_2, & \text{violation} = 0 \\ > P_2, & \text{violation} = h_m \end{cases} \quad \forall t \tag{25}$$

where h_m is the violation penalty for a capacity constraint, and P_2 is the capacity of a train.

From (7), the violation penalty for a set of consistency constraints is defined as

$$\sum_{l=1}^{L} y_{tl} \begin{cases} \leq x_t L_t, & \text{violation} = 0 \\ > x_t L_t, & \text{violation} = h_c \end{cases} \quad \forall t \tag{26}$$

where h_c is a violation penalty for a consistency constraint, and L_t is number of potential customers for timeslot t. From (9), the algorithm allocates a penalty if the assigned trains do not serve all customer demands. In other words, the covering constraint favours a combination of selected timeslots that covers all customers' bookings. The violation penalty within a set of covering constraints uses the same quantification, which is defined as

$$\sum_{t \in S_j} x_t \begin{cases} \geq 1, & \text{violation} = 0 \\ = 0, & \text{violation} = h_s \end{cases} \quad \forall t \tag{27}$$

where h_s is a violation penalty for a covering constraint.

4.2 Timeslot Weights

Timeslot violation h_t has been introduced artificially so that the search considers an estimation of total soft constraint violation, whilst some other hard constraints are still to be fully satisfied. The timeslot violation is regarded as if it were a hard violation until the capacity, consistency, and covering constraints have all been satisfied, then the timeslot violation is set to zero.

The algorithm assigns a penalty if a timeslot t is selected as a train departure time. This can be written as

$$x_t \begin{cases} = 1, & \text{violation} = h_t \\ = 0, & \text{violation} = 0 \end{cases} \quad \forall t \tag{28}$$

In contrast to (25)–(27) which imply a fixed penalty for each member of the associated set of constraints, a violation penalty for the timeslot violation h_t varies from timeslot to timeslot. The timeslot violation penalty depends on the possibility of assigning a particular timeslot on a train schedule with a minimum generalised cost.

An attempt to derive the timeslot violation in monetary units by trading off between the business criteria is not possible. This is because a train schedule is not a single timeslot, but is a set of the timeslots. Therefore, considering only a single timeslot separately from the others cannot represent a total cost for the rail carrier. However, as in practice some business criteria play a more important role than others, the relative weights for the criteria could be applied.

A rail carrier may assign a relative weight to the number of trains, customer satisfaction, and operating costs criteria in which the one with the lower weight is more important. In practice, given the relative weights 0.2, 0.5 and 0.3, the timeslot violation h_t is therefore obtained as

$$h_t = 0.2N_t + 0.5S_t + 0.3E_t \quad \forall t \tag{29}$$

where h_t is the timeslot violation if timeslot t is chosen (x_t = 1), N_t is the violation cost for the number of trains in timeslot t, S_t is the violation cost for the customer satisfaction in timeslot t, and E_t is the violation cost for the carrier's operating costs in timeslot t.

The violation cost N_t. We first assume that the higher the number of potential customers in timeslot t, the more likely that timeslot would lead to the minimum number of trains used. However, it is also necessary to consider the distribution of customer shipment size. Although there are a large number of potential customers in a timeslot, each customer shipment may be large. Therefore, such a timeslot could allow only a few customers to be served on a train so giving a high violation cost (or a priority) to this timeslot is no longer

reasonable. N_t is defined as

$$N_t = \frac{n_t}{n_{t(\max)}} \times 100 \quad \forall t \tag{30}$$

where $n_t = \frac{a_t}{\sum_{t=1}^{T} a_t} \quad \forall t$, with $a_t = \frac{\mu_t + \sigma_t}{C_t} \quad \forall t$, with μ_t the mean of customer shipment sizes in timeslot t, σ_t the standard deviation of the customer shipment sizes in timeslot t, and C_t the number of customers in timeslot t.

The violation cost S_t. Although the virtual loss of revenue in the generalised cost function could represent customer satisfaction in terms of a monetary unit, it is an indirect cost. In practice, the indirect cost is not obvious for rail expenditure as it affects the long-term financial plan. Therefore, in a competitive transport market, the direct cost that affects the short-term cash flow is regarded as more important. Since satisfaction probability represents customer satisfaction, we can sum up the satisfaction weight for the violation cost of each timeslot. S_t is defined as

$$S_t = \frac{s_t}{s_{t(\max)}} \times 100 \quad \forall t \tag{31}$$

where $s_t = \frac{b_t}{\sum_{t=1}^{T} b_t} \quad \forall t$, with $b_t = \frac{\sum_{t=1}^{T} W_t}{W_t} \quad \forall t$, and W_t a total customer satisfaction weight in timeslot t.

The violation cost E_t. A rail carrier may have different operating costs for different timeslots. The operating costs comprise train congestion cost and staff cost. Although a train schedule is a set of timeslots, we could consider E_t for the operating costs of each timeslot directly. This is because the operating cost is a cost unit and does not affect the number of timeslots in the optimal train schedule. The lower the operating costs for the timeslot, the higher the chance that the timeslot would lead to a schedule with the minimum generalised cost. E_t is defined as

$$E_t = \frac{e_t}{e_{t(\max)}} \times 100 \quad \forall t \tag{32}$$

where $e_t = \frac{U_t}{\sum_{t=1}^{T} U_t} \quad \forall t$.

5. COMPUTATIONAL RESULTS

The container rail scheduling model was tested on two sets of four successive weeks data from the eastern-line container service of the Royal State Railway of Thailand (SRT) and 184 shipping companies (customers). Each train has a capacity of 68 containers. The problem instances are summarised in Table 3.

Table 3. Problem instances (θ is a lower bound on number of trains).

Test case	Customer	Container	θ	SRT schedules Trains	SRT schedules Capacity	Supply–Demand Capacity	Supply–Demand Trains
W1	134	2907	43	57	3876	969	14
W2	116	2316	35	42	2856	540	7
W3	84	1370	21	28	1907	537	7
W4	109	2625	37	50	3400	775	13
W5	225	4115	61	73	4964	816	12
W6	198	3350	50	59	4012	612	9
W7	126	2542	38	49	3332	748	11
W8	286	4731	70	86	5848	1088	16

Table 4. Comparative results. The unit cost $\times 10^6$ Baht (Thai currency), OC: operating costs, VC: virtual loss of revenue, GC: generalised cost.

Test case	SRT cost	CBS cost OC	CBS cost VC	CBS cost GC	OC Reduction (%)
W1	5.17	4.51	1.05	5.56	12.77
W2	3.74	3.66	0.83	4.49	2.13
W3	2.19	1.86	0.33	2.19	15.07
W4	4.48	3.66	0.72	4.38	18.30
W5	6.49	5.88	2.18	8.06	9.40
W6	5.25	5.11	1.62	6.73	2.66
W7	4.66	4.25	0.87	5.12	8.79
W8	7.65	6.48	2.79	9.27	15.29

These eight instances were solved with the CBS algorithm described in Section 3.1 on a Pentium III 1.5 GHz. Each test case is run ten times using different random number seeds at the beginning of each run. If no improvement has been achieved within 2000 iterations, the search will terminate. For all test cases, we set $R = 20$, $(h_m, h_c, h_s) = 1, 1, 100$ respectively. Table 4 compares the model results with current practice.

Table 4 shows, in all the test cases, there are some reductions in terms of the number of trains and operating costs, but these are not considerable. This is because in practice the SRT schedule is not fixed at the same service level everyday. The rail carrier normally cuts down the number of train services with short notice if the train supply is a lot higher than the customer demand. This is done by delaying some customer's departure times according to its demand consolidation strategy.

Table 5. Results obtained by CBS and PCM.

Test case	CBS schedule			PCM schedule				
	Train	Cost*	Time	Train		Cost*		Time
	Avg.	Avg.	(s)	Avg.	±	Avg.	SD.	(s)
W1	51	5.56	230	47	2	4.18	0.56	105
W2	39	4.49	117	39	2	3.84	0.43	83
W3	24	2.19	74	24	1	2.09	0.27	61
W4	43	4.38	96	41	1	3.90	0.29	79
W5	70	8.06	882	66	3	7.02	0.72	351
W6	59	6.73	310	54	2	5.99	0.61	150
W7	46	5.12	145	43	1	4.93	0.25	92
W8	82	9.27	1170	75	3	8.15	0.66	509

* The generalised cost ($\times 10^6$ Baht, Thai currency)

However, the proposed model maximises customer satisfaction—in other words, minimises the virtual loss of future revenue within a generalised cost framework. Therefore, the schedule obtained by CBS could reflect the maximum degree of customer satisfaction with the minimum rail operating costs through a demand responsive schedule.

In addition, we demonstrate the performance of the constraint-based search incorporating the predictive choice model (PCM), and compare the results with CBS alone. The same test cases are run ten times using different random numbers at the beginning of each run. If no improvement has been achieved within 2000 iterations, the search will terminate. For all test cases, we set $R = 20$, $(h_m, h_c, h_s) = 1, 1, 100$ respectively, the number of flip trials $N = 20$, choice specific parameter $\beta_1 = 0.05$, decision method parameter $D = 75$, the number of fixing iterations $F = 100$, the number of fixing departure timeslots $x_t = 50$, the number of fixing selected booking timeslots ($y_{tj} = 1$) = 50, and the number of fixing unselected booking timeslots ($y_{tj} = 0$) = 200. The last three parameters govern the maximum number of variables which are allowed to be fixed at a point in the search and are necessary to allow some flexibility in the search. The results for the test cases are shown in Table 5.

Table 5 shows that the results of PCM are better than that of CBS alone. On average, the PCM schedule is 6.04% better in terms of the number of trains, and gives a reduction of 12.45% in generalised cost. Although in PCM learning from the search history implies a computational overhead over CBS, it is offset against a lower run-time required to find near optimal schedules, in particular for large test cases.

6. CONCLUSIONS

The ability to find a profitable train schedule along with satisfying customer demand using demand consolidation leads to some reductions in total operating costs, and enhances the level of customer service through demand responsive schedules.

The viability of using a CSP representation of the problem and solving this problem by CBS has been shown. CBS only relies on a simple local move and could accommodate new conditions without any change in the algorithm's structure. To achieve effective performance, problem-specific violation on a generalised scale is simply applied to constraints. The problem is first severely constrained so that few feasible solutions are likely to exist, the variables therefore would take consistent values in most of the feasible solutions. A new learning model is then introduced to predict a likely optimal value for those variables in order to help CBS target optimality in a probabilistic way. The predictive choice learning model is developed on theoretical grounds, using an analogy with discrete choice theory. The experimental results for the container rail service planning problem have demonstrated that CBS is a convenient and effective tool in producing good solutions, particularly when the predictive choice model is incorporated.

References

Aardal, K. (1998) Capacitated facility location: separation algorithm and computational experience, *Mathematical Programming*, **81**:149–175.

Abramson, D. and Randall, M. (1999) A simulated annealing code for general integer linear programs, *Annals of Operation Research*, **86**:3–24.

Anderson, P. S., Palma, A. and Thisse, J. (1992) *Discrete Choice Theory of Product Differentiation*, MIT Press, Princeton, NJ.

Arshad, F., Rhalibi, A. and Kelleher, G. (2000) *Information Management within Intermodal Transport Chain Scheduling*, European Project PISCES Report, Liverpool John Moores University.

Beasley, J. E. (1988) An algorithm for solving large capacitated warehouse location problems, *European Journal of Operational Research*, **33**:314–325.

Ben-Akiva, M. and Lerman, S. R. (1985) *Discrete Choice Analysis: Theory and Application to Predict Travel Demand*, MIT Press, Princeton, NJ.

Boffey, T. B. (1989) Location problems arising in computer networks, *Journal of the Operational Research Society*, **40**:347–354.

Connolly, D. (1992) General purpose simulated annealing, *Journal of the Operational Research Society*, **43**:495–505.

Cordeau, J., Toth, P. and Vigo, D. (1998) A survey of optimisation models for train routing and scheduling, *Transportation Science*, **32**:380–404.

Gomes, C. P., Selman, B. and Kautz, H. (1998) Boosting combinatorial search through randomisation. In *Proceeding of AAAI-98*, AAAI Press, Menlo Park, CA.

Gorman, M. F. (1998) An application of genetic and tabu searches to the freight railroad operating plan problem, *Annals of Operations Research*, **78**:51–69.

Henz, M., Lim, Y. F., Lua, S.C., Shi, X. P., Walser, J. P. and Yap, R. (2000) Solving hierarchical constraints over finite domains. In *Proceedings of the 6th International Symposium on Artificial Intelligence and Mathematics* (Fort Lauderdale, FL).

Hansen, E.R. (1992) *Global Optimisation Using Interval Analysis*, Dekker, New York.

Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B. and Chickering, M. (2001) A Bayesian approach to tackling hard computational problems. In *Proceedings of UAI01*, pp. 235–244.

Huntley, C. L., Brown, D. E., Sappington, D. E. and Markowicz, B. P. (1995) Freight routing and scheduling at CSX transportation, *Interfaces*, **25**:58–71.

ILOG (2000) ILOG OPL Studio Version 3.5.1 Reference Manual, ILOG Inc.

Indra-Payoong, N., Srisurapanon, V. and Laosirihongthong, T. (1998) Factors influencing modal choice for freight transportation. In *Proceedings of the Civil and Environmental Engineering Conference, New Frontiers and Challenges (Bangkok, Thailand)*, 419–26.

Johnson, R. and Wichern, D. (1996) *Applied Multivariate Statistical Analysis*, Cambridge University Press, Cambridge.

Kallenberg, O. (1997) *Foundations of Modern Probability*, Springer, New York.

Kearfott, R. B. (1998) On proving existence of feasible points in equality constrained optimisation problems, *Mathematical Programming*, **83**:89–100.

Kochmann, G. A. and McCallum, C. J. (1981) Facility location models for planning a transatlantic communications network, *European Journal of Operational Research*, **6**:205–211.

Kraft, E. R. (2002) Scheduling railway freight delivery appointments using a bid price approach, *Transportation Research*, **36A**:145–165.

Larraaga, P. and Lozano, J. A. (2002) *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer, Dordrecht.

Lau H. C., Lim, Y. F. and Liu, Q. (2001) Diversification of neighbourhood via constraint-based local search and its application to VRPTW. In *Proceedings of CP-AI-OR*, pp. 361–374.

Marder, E. (1997) *The Laws of Choice: Predicting Customer Behavior*, The Free Press, Simon and Schuster, New York.

Marin, A. and Salmeron, J.(1996) Tactical design of rail freight networks. Part II: local search methods with statistical analysis, *European Journal of Operational Research*, **94**:43–53.

McAllester, D., Selman, B. and Kautz, H. (1997) Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pp. 459–465.

Ministry of Commerce (2002) *Thai Commodity Market Price*, Department of Business Economics, Ministry of Commerce, Thailand.

Newman, A. M. and Yano, C. A. (2000) Scheduling direct and indirect trains and containers in an intermodal setting, *Transportation Science*, **34**:256–270.

Nonobe, K. and Ibaraki, T. (1998) A tabu search approach to constraint satisfaction problem as a general problem solver, *European Journal of Operational Research*, **106**:599–623.

Parkes, A. and Walser, J. (1996) Tuning local search for satisfiability testing. In *Proceedings of AAAI-96*, pp. 356–362.

Patrick, R. (1982) Algorithm AS 181: The W test for normality, *Applied Statistics*, **31**:176–180.

Proll, L. and Smith, B. (1998) ILP and constraint programming approaches to a template design problem, *INFORMS Journal of Computing*, **10**:265–277.

Resende, G. C. and Ribero, C. C. (2001) *Greedy Randomized Adaptive Search Procedures, State-of-the Art Handbook in Metaheuristics*, F. Glover and G. Kochenberger (Eds.), Kluwer, Dordrecht.

Selman, B., Levesque, H. and Mitchell D. (1992) A new method for solving hard satisfiability problems. In *Proceedings of AAAI-92*, pp. 440–446.

Selman, B., Kautz, H. and Cohen, B. (1994) Noise strategies for improving local search. In *Proceedings of AAAI-94*, pp. 337–343.

Shapiro, S. S. and Wilk, M. B. (1965) An analysis of variance test for normality (complete samples), *Biometrika*, **52**:591–611.

Smith, B., Stergiou, K. and Walsh, T. (2000) Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings of AAAI-2000*, Austin, TX.

Trotter, H. F. (1959) An elementary proof of the central limit theorem, *Archives of Mathematics*, **10**:226–234.

Walser, J. P. (1999) *Integer Optimisation by Local Search: A Domain-Independent Approach*, Lecture Notes in Artificial Intelligence, Vol. 1636, Springer, Berlin.

Yano, C. A. and Newman, A. M. (2001) Scheduling trains and containers with due dates and dynamic arrivals, *Transportation Science*, **35**:181–191.

RULE-BASED SYSTEM FOR PLATFORM ASSIGNMENT IN BUS STATIONS

B. Adenso-Díaz
Engineering School, Universidad de Oviedo, Spain

Abstract Of all the tasks related to bus station management, one of the most important decisions that greatly affects the quality of the service is the assignment of platforms. A bad assignment procedure may be uncomfortable for frequent passengers who often suffer changes in departure gates, or may even be affected by the lack of platforms. Assignment is complicated due to the frequent incidents that cause delays in or the moving forward of services (with the corresponding changes in assignment), and to increases in services that convert the platforms into a limited resource. Even though this problem of platform assignment has been studied in more detail in the field of airport terminals, the difficulties in road transportation are both distinct and significant. In this paper we develop an intelligent system employing rule-based models for the daily management of the platforms of a bus station. The developed system is currently being used in a real bus station with more than 1,000 daily services.

Keywords: gate assignment, decision support systems, expert systems, bus-station management.

1. INTRODUCTION

Within the transport sector, one of the most widespread decisions that must be taken by those in charge of stations or airports is that of assigning the gate or platform from which each of the offered services must leave. The chosen option is highly relevant in relation to the quality of the service offered, since sub-optimum assignments give rise to dissatisfaction and protests on the part of customers (who must proceed unnecessarily from one part of the station to another), protests from transport companies (which may suffer delays as a result of congestion caused by the lack of correct assignments), etc.

This paper describes a platform assignment system designed for managing bus stations. These types of models have been mainly designed so far for airports in an attempt to use gates efficiently (good distribution among companies, minimisation of delays, etc.) Since the mid-1980s (Hamzawi, 1986, presents one of the first results for this problem using microcomputers), differ-

ent gate assignment algorithms have been proposed for airport terminals, with real cases such as Singapore airport (Cheng, 1997) or Taiwan (Yan and Chang, 1998).

The basic goal in all these cases is to minimise the walk distance of connecting passengers (or baggage collection) at the terminal, thus reducing connecting times. Some authors (for instance Yan and Huo, 2001) have incorporated passenger waiting time, defining multi-criteria approaches.

Two different approaches have been considered in the literature for solving the gate assignment problem: *expert systems and simulation* (Srihari and Muthukrishnan, 1991, present a review of the use of ES; Gosling, 1990, introduced one of the first applications in this field, inspiring later papers such as Su and Srihari, 1993; Yan *et al.*, 2002, present a simulation framework tested in Taipei airport able to analyse the effects of stochastic flight delays); and *exact procedures* (such as Mangoubi and Mathaisel, 1985, using linear programming; Haghani and Chen, 1998, and Bolat, 2000, using a QAP with temporal constraints, or Yan and Chang, 1998, using network models). Cheng (1997) presents a third alternative (considered here), a hybrid rule-based system and a heuristic assignment procedure.

As far as we know, the gate assignment problem has not yet been studied in the bus management case, an environment with distinct requirements and constraints. The goal of reducing the distance passengers walk becomes secondary, since the size of stations is in general much smaller than that of airports.

However, several new criteria emerge that the station manager must take into consideration: services with destinations in the same geographical area or offered by the same company are usually placed close to one another so that frequent passengers can know their way around the station better and so that the employees of each carrier are near to the place they have to carry out their functions; when a high-demand service is covered by several buses, these must leave in a staggered fashion; if possible, on long-distance services where passengers usually carry luggage, it would be desirable for adjacent platforms to be empty at arrival and departure times so as to avoid a pile-up of luggage on platforms which are normally narrow; not all platforms are valid for all sizes of buses; some platforms must be reserved as temporary parking for the buses which depart some time after arriving at the station; and so on.

All this must be considered, together with endeavouring (as occurred in the air transport sector) to get high demand services to arrive at and depart from platforms near to the ticket offices or main doors, and above all attempting to avoid any delay due to a shortage of platforms at the time of offering the service.

In the case of buses, the choice of criteria is both complex as well as offering a multiplicity of options. This, together with the need to use more user-friendly

tools in view of the generally lower technological level of the bus sector in comparison with that of the aeronautic sector, leads to the consideration of approaches of the third type mentioned above: that is, approaches based on rules that may be dynamically chosen by the person responsible for defining assignments on the basis of the criteria that he or she wishes to take into account at the time.

This paper presents the solution given to the problem for assigning platforms in a newly built bus station in the North of Spain. Section 2 presents the framework within which the new system was developed and the architecture chosen for the solution. Section 3 describes the functioning of the first of the two modules into which the system was divided, and the algorithm defined to solve it. Section 4 describes the on-line characteristics of the module.

2. ARCHITECTURE OF THE SYSTEM

The new bus station of the city of Oviedo, the capital of the Principality of Asturias, in the North of Spain, was inaugurated in 2003. The new station handles 1,100 bus services per day, and has 44 platforms. It should be noted that not all the platforms are valid for any type of bus, nor do they all have the same departure and entry load if the constraints that normal daily operations impose are respected. One part of their management consists in deciding which platform each offered service is to arrive at and leave from.

To do so, the station management asks the different companies that operate there about their needs and preferences as regards platforms and, in view of the requests of all of these operators, must make an assignment that will be operative on any particular day, and which is announced to passengers via information screens.

There are two sources of difficulty when carrying out a valid assignment on any particular day. Firstly, the one underlying the theoretical assignment that might initially be constructed in an attempt to satisfy the carriers' requests, and secondly the one derived from the day to day alterations imposed on this theoretical plan by delays and unforeseen events that mean that re-assignments must be made with respect to this theoretical plan. As a result, the system designed here was structured in two modules (Figure 1):

- *Off-line module.* This module generates generic assignment plans for the bus station, valid for different periods with common specific timetables (summer, Easter, summer Fridays, etc). The output provides valid assignments generated using different rules ready to be chosen for a specific set of days. All these assignments make up a database with different options (one for each set of chosen rules) valid for different types of days (timetable of services). A typical couple of dozens of plans are usually stored in this data base.

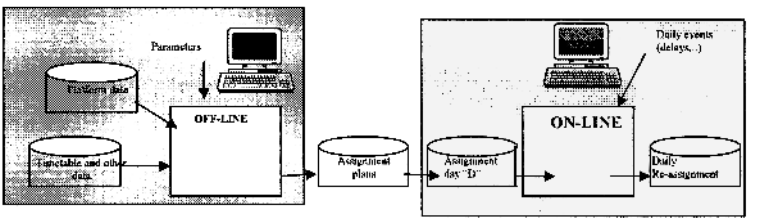


Figure 1. Structure of the overall system for platform assignment.

- *On-line module.* Once one of the generic plans generated in the off-line module and available in the corresponding database has been chosen for a specific day, the daily operation at the bus station starts. It is quite normal for unplanned events to occur (delays, an increase in the number of buses in a service, etc.) which necessitate the changing of some of the preset assignments. The re-scheduling is performed by this module, taking into consideration the defined constraints and the current configuration of the station.

3. OFF-LINE MODULE

As mentioned above, the goal of this module consists in generating an assignment plan for a specific group of days with services in common, considering a set of rules defined by the station manager for these days. Therefore, an important step in this module is to identify which rules the manager wishes to select for the assignment to fit these rules.

There are two types of rules that are defined by the manager for constructing the assignment: direct rules (DR) and approximate rules (AR). The former indicate some kind of "hard constraints" that must be satisfied by the solution. An example could be "Buses going to Madrid must depart from platforms $4, 5, \ldots, 15$" (denoting a preference), or "Buses coming from Madrid must not arrive at platforms 1, 2 or 3" (denoting an exclusion), or "Company X must not share platforms with company Y" (avoiding assignments in the case of another assignment having been made previously).

Many different types of rules may be composed with this syntax, as platform preferences may be chosen depending on destination or origin, geographic areas, companies, sizes of buses, or temporary parking areas may also be chosen. A weighting ranked $w_r \in \{1, \ldots, 9\}$ is associated with each rule that indicates the importance given by the manager to this rule's specifications being complied with (the higher the weighting, the more important it is).

The approximate rules, on the other hand, are of the fuzzy type (soft constraints) and are built with close/far relationships. Typical examples could be

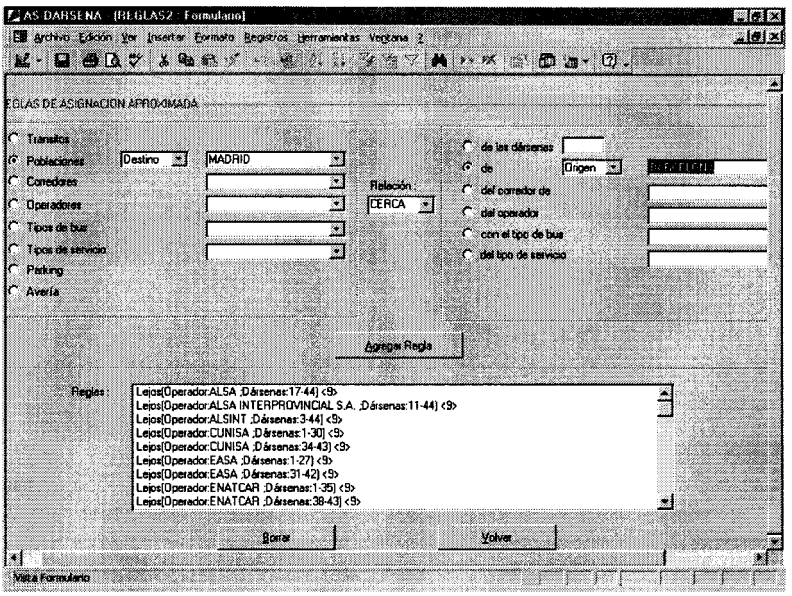


Figure 2. Example of a screen where approximate assignment rules are defined.

"Departure to Madrid must be close to arrivals from Barcelona" (see example in Figure 2) or "Platforms used by company X must be far from platforms used by company Y". Here the rules may be of the close/far type between different concepts (arrivals, departures, companies, etc) or in relation to certain platforms. In the former case, the relation is established with respect to other previous assignments and is therefore a dynamic dependence that varies in accordance with the assignment history that is being constructed. In the latter, however, the relationship is established with regard to fixed positions (platforms). We call these last rules ARPs (Approximate Rules associated with Platforms) and they may be considered from the beginning in the assignment algorithm, as they do not depend on the previously performed assignment sequence.

A TH threshold must be defined for processing the close/far rules (depending on the layout of the station) that indicates the distance between two platforms above which they cannot be said to be close, and below which it is understood that they are not far (Figure 3). The distance between two platforms, when the layout of the station is approximately linear, could be simply defined as $d_{ij} = |i - j|$.

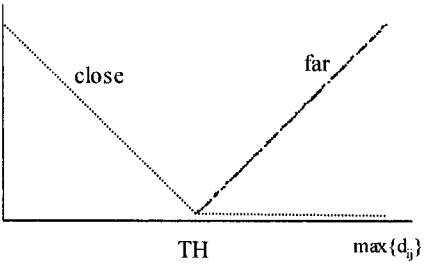


Figure 3. Membership function for the variables "close" and "far" related to the distance between platforms.

A third group of rules are those that indicate the desire to keep platforms next to those with passengers that will forseeably be carrying a lot of luggage (for example long-distance services) unassigned.

To avoid introducing rules that are inconsistent with previously defined ones, any new rule is checked against those introduced previously before it is accepted. This new rule will then only be accepted if it does not produce any inconsistencies.

3.1 Approach for Assigning Platforms to Services

Services are sorted according to the number of alternative platforms available, taking into consideration the direct rules and the timetable incompatibilities found so far. The service with fewer alternatives will be the first one assigned to a platform, and—given that all these alternatives already satisfy the hard constraints—approximate rules are then used to determine the most interesting alternative from among these. The idea of first scheduling the items with the least number of available alternatives (LSD, least saturation degree) is successfully used in graph colouring algorithms (Brelaz, 1979), and its logic is easily translated into our problem.

A data structure was created to generate a solution that, in line with this idea, assigns a platform to each bus while respecting the introduced rules as far as possible. Figure 4 presents a schematic representation of this structure, in which each row represents a bus to which a platform must be assigned.

The first column indicates the service that the bus will perform and the second the number of buses that will carry out this service. The third column contains a "Status" array for each bus j, with the same number of positions as platforms in the station. Each position i may contain: a number 1–9 (the priority assigned to platform i for service j by a direct assignment rule); the number 0 (if no rule affects this platform for service j); the character "-" (if platform i is excluded from possible assignment to j due to incompatibility with a rule

SERVICE	#bus	STATUS	#"-" in STATE	CAPR	LAPR	Assign ment
Oviedo-Barcelona S1	1	[0,-,-,6,6,0,P,9,...]	2	[0,2,2,3,3,0,9,9,...]	[0,0,0,0,0,7,7,0,...]	
Oviedo-Madrid S2	1	[0,-,-,-,3,0,P,0,...]	3	[0,0,0,5,5,5,0,0,...]	[0,2,2,3,3,9,9,9,...]	
Gijón-León S3	1	[0,0,5,5,5,0,P,7,...]	0	[0,0,0,3,3,0,9,...]	[9,0,0,0,0,0,0,0,...]	
Gijón-León S3	2	[0,0,5,5,5,0,P,7,...]	1	[0,0,0,3,3,0,9,...]	[9,0,0,0,0,0,0,0,...]	
Oviedo-Noreña S4	1	[0,-,-,-,-,0,P,9,...]	4	[0,6,1,1,3,0,0,0,...]	[0,0,0,0,0,0,6,6,...]	**8**

Figure 4. Scheme of the data structure used for the assignment of platform in the off-line module.

or because it is already assigned to another service $k \neq j$ at the required time). The character "P" indicates that platform i is assigned to parking, and so is also excluded from assignment to service j.

The fourth column indicates how many platforms are excluded from assignment to this service (i.e. the number of "-" in Status). This data will be used to rank the assignment priority of the services according to LSD logic.

The fifth (Capr) and sixth (Lapr) columns gather information about the approximate rules of the close and far type, respectively. As in the case of Status, a number in position i of one of these arrays would indicate that an approximate rule is conferring this weighting to platform i for this service. For example, in Figure 4, once the Oviedo-Noreña service has been assigned to platform 8, the "*Close rule*" (Destination: León; Destination: Noreña) with a weighting of 9 has forced the introduction of the value 9 in the 8th position of the Capr array, corresponding to two buses whose destination is León (rows 3 and 4).

The seventh column contains the final assignment carried out, and once all the rows have been completed, the complete solution will have been obtained.

Working on this data structure, the assignment algorithm constructs the final solution (Figure 5). In the first three steps, the arrays are initialised according to the DR and ARP rules. In Step 4, DR rules are eliminated if they give rise to problem infeasibility. Step 5 is a pre-selection of the services S that have a greater likelihood of being chosen in the current iteration so as to then search for their assignment (LSD rule), taking into account the fact that a large cardinal number in the set S in this step would slow down the entire calculating process in the two following steps.

A. Prepare data

1. STATE=CAPR=LAPR='0'
2. ∀rule r∈DR, update STATE with "P", "-", or assigning its weight
3. ∀rule r∈ARP, update CAPR & LAPR

B. Iterating for assignment

4. If ∃service S_i: ∀j, $STATE_j$='-', drop the rule r∈DA with lowest weight. Go back to step 1
5. **Pre-select a set S** of services S_i ready for immediate assignment:
   S is made up of S_i's with the highest value in column "# '-' in STATE" (rule LSD)
6. **Pre-calculate** $\forall S_i \in S$ the **best assignment** d_k^i.
   Let D the set of possible assignments to S_i, considering STATE
   d_k^i=arg max $\{f_{apr}(d_m, CAPR, LAPR)\}$ $\forall d_m \in D$
   Ties are broken choosing the d_m less busy before the departure time
7. **Choose the** $S_i \in S$ **to be assigned now**, and assign d_k^i to it.
   Select S_i=arg min$\{g(S_j, d_k^j, STATE)\}$ measuring function g the number of "-" induced in the other services in case the assignment $<S_j, d_k^j>$ is made.
8. Update STATE; ∀r∈AR\ARP affecting last assignment, update CAPR & LAPR. Go back to step 4 in case there are services not assigned yet.

C. Output results

9. Output #rules r∈DA discarded in step 4; and #rules r∈DA that do not hold
10. Output a measure of the fitness *F* of assignments, according to r∈AR

Figure 5. Algorithm for producing a solution according to the preset rules.

In Step 6, the best assignment according to AR rules is chosen for each pre-selected service using an f_{apr} function defined as

$$\begin{aligned} & f_{apr}(d_m, \text{Capr}, \text{Lapr}) \\ & \quad = \text{AVERAGE}_{\forall j:\text{Caprj}\neq 0}[\text{Capr}_j \cdot \max\{-\text{TH}; \text{TH} - |d_k - j|\}] \\ & \qquad + \text{AVERAGE}_{\forall j:\text{Laprj}\neq 0}[\text{Lapr}_j \cdot \min\{\text{TH}; |d_k - j| - \text{TH}\}] \end{aligned} \quad (1)$$

that returns a greater value to platform d_k depending on whether there are Capr_i weightings in positions close to k, or whether the Lapr_i weightings of position k are far.

In Step 7, the assignment that is effectively made in this iteration is chosen and the arrays are updated in Step 8 to then continue iterating until all the services are assigned.

Once completed, the final assignment is displayed as well as an assessment of the quality of the solution found (Figure 6). This assessment is evaluated (Step 10) using the function F that measures the fitness of the solution in rela-

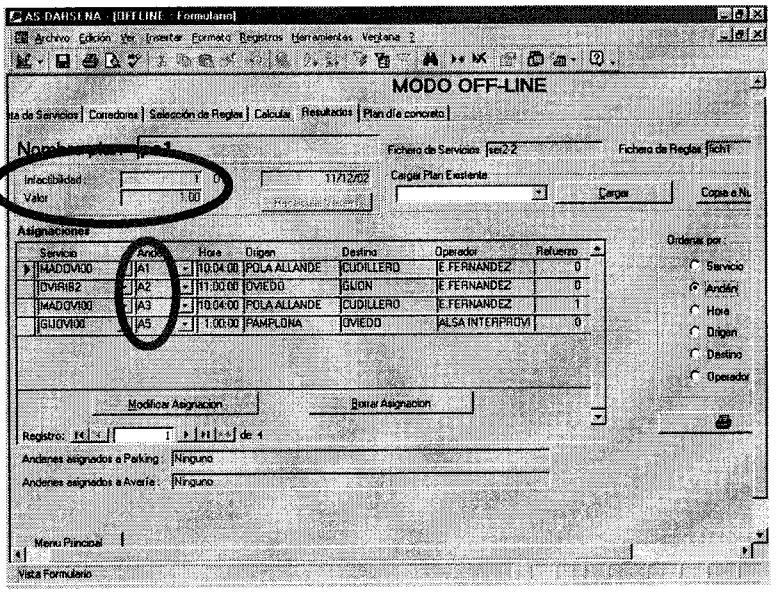


Figure 6. Screen with the final assignments (centre) and the assessment of their quality (upper-left). As usual, it is possible to introduce manual corrections in the case of the manager wishing to consider additional constraints.

tion to the fuzzy rules of type "close/far":

$$F = \frac{\sum_{\forall r \notin DR} w_r \cdot \underset{\text{Service } s \text{ affected by } r}{Average} \{a_{sr}\}}{\sum_{\forall r \notin DR} w_r} \tag{2}$$

$a_{sr} \in [0,1]$ being a measure of the fitness of the assignment made to service s in relation to rule $r \in$ AR. If we call k the platform assigned to s, and the closest assigned platform affected by rule r is called d, a definition of a_{sr} for rules r related to "close" is given by (3), and for "far" rules by (4):

$$a_{sr} = \begin{cases} 1 & \text{if } |d-k| \leq 1 \\ 1 + \dfrac{1-|d-k|}{\mathrm{TH}-1} & \text{if } 1 < |d-k| < \mathrm{TH} \\ 0 & \text{if } |d-k| \geq \mathrm{TH} \end{cases} \tag{3}$$

$$a_{sr} = \begin{cases} 0 & \text{if } |d-k| \leq \mathrm{TH} \\ \dfrac{|d-k|}{\mathrm{TH}} - 1 & \text{if } \mathrm{TH} < |\mathrm{d}-\mathrm{k}| < 2 \cdot \mathrm{TH} \\ 1 & \text{if } |d-k| \geq 2 \cdot \mathrm{TH} \end{cases} \tag{4}$$

Although the calculation time depends to a great extent on the number of services and the number and types of rules introduced, in the station where it is being tested the system takes 8 minutes on a Pentium III to generate a solution for 1,100 services using some 50 rules.

4. ON-LINE MODULE

On a normal working day at the station, one of the off-line plans generated in the above module in accordance with different scenarios is initially loaded. During the day, however, two types of event may take place that force the pre-established assignment to be modified:

- Incidents that alter the arrival or departure of buses (leading, as a result of a delay, to a platform that is assigned to a service currently being occupied by the following service assigned to it, with the consequent need to find a new assignment for the first service) or the number of buses (since, owing to passenger demand, it has been decided to reinforce a service with another bus, for which a new platform will have to be found. In these cases, modifications will have to be made to the previously realised assignment for the service.

- The creation of new special services not contemplated in the standard off-line plan chosen for the time of year or the day under consideration. In this case, a totally new assignment will have to be made for the service.

In both cases, the rules that the manager wishes to consider when making the re-assignment are chosen via a new screen. In the first case, the rules are of the type "find the closest platform possible to the pre-assigned platform for this service", or to another specific platform that is established. In the second case, rules are used that search among the platforms that are free at the time the service has to be offered for those that are closest either to those where other similar services to the new one leave from or arrive at, or those that belong to the same company, etc.

The algorithm of this module is simpler to implement than the above off-line algorithm. As the off-line plan is fixed and the time horizon we are dealing with is as close as it is in this case, being affected by only one service, the possible re-assignment alternatives are very few in number.

5. CONCLUSIONS

Daily management of a bus station introduces many different types of fuzzy and hard constraints when assigning platforms to services. A complete system has been designed for solving the platform assignment problem in bus stations

based on the definition by the management of a set of rules that the assignment solution should satisfy.

Two different types of rules were considered for the off-line module. Based on their characteristics, a search procedure looks for a solution satisfying (if possible) all the hard constraints, maximising the value considering the approximate rules. The on-line module searches for an alternative assignment when, in the day to day use of the system, any unplanned event occurs that forces re-assignment of platforms. This logic could be easily transferred to other environments where complex rules, which make the use of exact approaches difficult, have to be defined for the assignment of resources.

The system is currently being used at Oviedo bus station (Northern Spain), a facility with more than 1,100 daily bus services. The computation time needed for the off-line calculation in this station is around 10 minutes, being the system able to find solutions satisfying the current constraints established at the site.

References

Bolat, A. (2000) Procedures for providing robust gate assignments for arriving aircrafts. *European Journal of Operational Research*, **120**:63–80.

Brelaz, D. (1979) New methods to color the vertices of a graph. *Communications of ACM*, **22**:251–256.

Cheng, Yu (1997) A knowledge-based airport gate assignment system integrated with mathematical programming. *Computers and Industrial Engineering*, **32**:837–852.

Gosling, G. D. (1990) Design of an expert system for aircraft gate assignment. *Transportation Research* A, **24**:59–69.

Haghani, A. and Chen, M.-Ch (1998) Optimizing gate assignments at airport terminals. *Transportation Research* A, **32**:437–454.

Hamzawi, S. G. (1986) Management and planning of airport gate capacity: a microcomputer-based gate assignment simulation model. *Transportation Planning and Technology*, **11**:189–202.

Mangoubi, R. S. and Mathaisel, F. X. (1985) Optimizing gate Assignments at airport terminals. *Transportation Science*, **19**:173–188.

Srihari, K. and Muthukrishnan, R. (1991) An expert system methodology for aircraft-gate assignment. *Computers and Industrial Engineering*, **21**:101–105.

Su, Y. Y. and Srihari, K. (1993) A knowledge based aircraft-gate assignment advisor. *Computers and Industrial Engineering*, **25**:123–126.

Yan, S. and Chang, C.-M. (1998) A network model for gate assignment. *Journal of Advanced Transportation*, **32**:176–189.

Yan, S. and Huo, C.-M. (2001) Optimization of multiple objective gate assignments. *Transportation Research* A, **35**:413–432.

Yan, S., Shieh, C.-Y. and Chen, M. (2002) A simulation framework for evaluating airport gate assignments. *Transportation Research* A, **36**:885–898.

MEASURING THE ROBUSTNESS OF AIRLINE FLEET SCHEDULES

F. Bian[1], E. K. Burke[2], S. Jain[3], G. Kendall[2], G. M. Koole[4], J. D. Landa Silva[2], J. Mulder[5], M. C. E. Paelinck[5], C. Reeves[6], I. Rusdi[7], M. O. Suleman[1]

[1]*University of Oxford, UK*
[2]*The University of Nottingham, UK*
[3]*Aston University, UK*
[4]*Free University of Amsterdam, The Netherlands*
[5]*KLM Airlines, The Netherlands*
[6]*Coventry University, UK*
[7]*Technical University of Delft, The Netherlands*

Abstract Constructing good quality fleet schedules is essential for an airline to operate in an effective and efficient way in order to accomplish high levels of consumer satisfaction and to maximise profits. The robustness of an airline schedule is an indicative measure of how good the schedule is because a robust plan allows the airline to cope with the unexpected disturbances which normally occur on a daily basis. This paper describes a method to measure the robustness of schedules for aircraft fleet scheduling within KLM Airlines. The method is based on the "Aircraft on Ground (ACOG)" measure, it employs statistical methods (although alternative methods were also considered) and it is shown to provide a good estimation of the robustness of a given schedule.

Keywords: modelling, airline scheduling, schedule quality measures.

1. INTRODUCTION

The problem of generating fleet schedules is crucially important to the efficiency of an airline (Barnhart *et al.*, 1997; Barnhart and Talluri, 1997). An effective schedule can lead to significant savings. It can also, and perhaps more importantly, contribute to higher levels of customer satisfaction. Customers who experience regular delays with a particular airline are likely to take their custom elsewhere. Of course, delays are inevitable for a wide range of reasons (e.g. technical breakdowns, security alerts, adverse weather, etc). However, an indicative measure of the quality of an airline schedule is its level of *robustness*: How well can a schedule cope with a delay(s) to a particular aircraft(s)?

Is there enough slack in the schedule to minimise the knock on effect of a delay to a particular aircraft? If there is no slack in the schedule then a delay to one aircraft could affect a significant proportion of the fleet and this could have major resource implications. If passengers miss connecting flights then the airline has to cover the incurred costs. However, building slack into the schedule is expensive. It essentially involves aircraft standing idle. One of the goals in trying to generate a high quality fleet schedule is to build in enough slack to ensure that the schedule has an acceptable level of robustness while, at the same time, attempting to keep costs at an effective level. It would be very easy indeed to build a very robust schedule. However, it would be too expensive to implement. It would also be possible to build a schedule which minimises cost by decreasing aircraft idle time. However, this could easily lead to an increase in the overall incurred costs if one minor delay to one aircraft leads to a chain of delays. In summary, the goal is to provide an effective balance between robustness and aircraft idle time.

The integration of schedule optimisation algorithms and other systems in an airline is crucial to achieve an effective scheduling environment that considers all functions of the airline (Mathaisel, 1997). Reviews of research on airline scheduling are presented in Etschmaier and Mathaisel (1985) and Richter (1989). A more recent survey on models and solution methods for a range of problems in aircraft scheduling was carried out by Gopalan and Talluri (1998).

Aircraft scheduling is often addressed simultaneously with other associated problems. An example is provided by fleet assignment with time windows where the assignment of aircraft is carried out simultaneously to scheduling flight departures in order to improve flight connection opportunities and minimise costs (Rexing *et al.*, 2000). The scheduling of maintenance operations and of aircraft are considered simultaneously using network models and a two-phase heuristic by Feo and Bard (1989), while crew availability and maintenance operations are taken into account while tackling the fleet assignment problem in Clarke and Hane (2001). The additional constraint of equal aircraft utilisation when tackling fleet assignment and aircraft routing problems is considered by(Barnhart *et al.* (1998). A network model for large-scale fleet assignment problems that permits the expression of constraints within a unified framework was presented by Rushmeier and Kontogiorgis (1997).

Integer linear programming techniques have been applied by several researchers to tackle fleet assignment, aircraft routing and related problems (Abara, 1989; Subramanian, 1994; Hane *et al.*, 1995). Dynamic programming and heuristics have also been investigated for the problem of fleet assignment (El Moudani and Mora-Camino, 2000). Recently, meta-heuristic methods have been used to tackle airline scheduling problems. For example, simulated annealing was applied to the optimisation of airline schedules by Mashford and Marksjo (2001). Sosnowska and Rolim (2001) showed that by applying sim-

ulated annealing to the fleet assignment and aircraft routing, improvements of about 10–20% over the method used by the company could be achieved. A genetic algorithm was applied to generate alternative routes for air traffic by Oussedik *et al.* (2000). Also recently, genetic search methods have been applied to solve the problem of sequencing the arrival of aircraft in airports (Hansen, 2004; Ciesielski and Scerri, 1997, 1998).

Re-scheduling is a crucial activity for airlines and it has to be carried out on a daily basis due to a number of uncertainties and unforeseen events. Disruptions of planned schedules can result in a chain of events that can cause major disruptions throughout the system. A survey of techniques employed to recover from these disruptions is presented by Filar *et al.* (2001). A stochastic model is employed by Rosenberger *et al.* (2003) to show that the actual performance of an airline differs greatly from the planned performance while Argüello and Bard (1997) propose a GRASP method to reconstruct schedules while minimising costs and satisfying constraints. Network models and Lagrangian relaxation were used by Yan and Lin (1997) for aircraft re-scheduling given a specific disruption that affects the airline operations greatly and causes substantial decrements in profits and levels of service: the temporary closure of airports (see also Thengvall *et al.*, 2001, 2004). The problem of changing the assigned aircraft to specific flights while satisfying existing constraints is addressed by Jarrah (2000), Talluri (1996) and Klincewicz and Rosenwein (1995). A steepest ascent local search heuristic was applied by Love *et al.* (2002) to re-schedule aircraft and it was capable of finding good quality schedules in a short amount of time.

The problem that is addressed in this paper is discussed in the next section. It represents a real-world problem that faces KLM Airlines on a daily basis.

2. PROBLEM DESCRIPTION

Within KLM, two departments are responsible for the fleet schedule. The network planning department produces schedules which are then passed to the operations department who has the responsibility for implementing them and running them on a day-to-day basis. These two departments have conflicting objectives. The network department aims to produce a schedule which is as cost effective as possible. This essentially means maximising aircraft usage by minimising their idle time. The operations department prefers schedules that have enough slack to ensure a certain level of robustness. This means having as much aircraft idle time as possible. Then, the overall aim is to produce a schedule with the right balance between these two conflicting objectives briefly described above.

The aim for KLM is to introduce a method that checks the robustness of a schedule, from the network department, before it is passed to the operations

department for implementation. One way to achieve this is to run a simulation. However, this is seen as too time consuming and other methods are sought to test for the robustness of the schedule.

KLM flies to over 150 destinations using 97 aircraft. Four times a year, a new flight schedule is developed. Though the operational feasibility is taken into account to a certain degree during the development process, the aim at that stage is largely to maximise the number of seats that can be sold. During schedule development, KLM considers various commercial aspects such as the expected demand per destination and the number of possible transfer connections at Schiphol Airport in Amsterdam.

The realisation of a flight schedule involves a number of parties. As described above, the initial plan is developed by KLM's network planning department. The initial plan is based on commercial and strategic insights and long term plans for the fleet composition, cabin crew and baggage handling.

Two months before the beginning of a schedule plan, the plan is handed over to the operational department, the Operation Control Centre. From that moment on they are the owners of the plan and small adaptations have to be evaluated and approved by them. This department will try to prevent and solve problems such as emergencies and bottlenecks and, in case of unsolved problems, try to minimize the effects on succeeding flights. A final plan is created two weeks before the beginning of the plan where passenger bookings are matched with aircraft capacities

In order to monitor the performance of a flight schedule, some critical performance indicators are defined. These are

- The departure and arrival punctuality: that is, the percentage of flights that departed or arrived on time.
- The completion factor: that is, the percentage of accomplished flights. These are all flights that were not cancelled.
- The No Connection Passenger factor: that is, the percentage of transfer passengers that missed their connections due to operational problems.
- The Irregularity-rate: that is, the number of bags that were not delivered on time.

For the punctuality performance indicator the contribution of each of the involved parties is also monitored. This introduces the concept of building blocks. The whole operational process is divided into sub processes, (the so called building blocks). Each building block is owned by a *capacity and service provider*, these being Ground Services, Front Office, Air Traffic Management, Engineering and Maintenance, Cabin and Cockpit Crew, Cargo and Operations Control. Seven Building Blocks have been established, these are called

- BB1: Flight
- BB2: Arriving aircraft
- BB3: Layover aircraft
- BB4: Departing aircraft
- BB5: BB5.1 Transferring passengers
  BB5.2 Transferring baggage
- BB6: BB6.1 Arriving passengers
  BB6.2 Arriving baggage
- BB7: BB7.1 Departing passengers
  BB7.2 Departing baggage.

A diagrammatical representation of the temporal sequence of the building blocks and their relationships to each other is shown in Figure 1. These have been delimited in order to provide clear process distinction as well as accountability.

The doors being opened and closed are the points at which responsibility passes from one capacity and service provider to another. The distinction of the *first door* being opened is made because a door can either be the passenger door(s) or a baggage door(s). For example, once a plane has physically landed it is not actually considered to have *landed* (i.e. with responsibility passed to the ground staff) until one (passenger OR baggage) door has been opened. In contrast, responsibility changes back again when *all* doors have been closed, not just one door.

All these agreements and the flight schedule itself comes together into an operational plan. This functions as a contract between Network, Operations Control and the Building Blocks (Capacity & Service Providers). The plan covers an operational plan period of between 2 to 4 months spread over the year. It consists of agreements concerning a schedule plan and a capacity plan position for each specific period. It contains a demand-driven schedule that has been fully checked with the Building Block representatives (Capacity & Service Providers) and Operations Control by means of an operational check. Eventually the agreements enable each provider to deliver an operational performance forecast. This could deviate from the targets as laid down in the corresponding Business Plan. Each operational plan will be finalized two months (at the latest) prior to each operational plan period.

The schedule is usually published as an Aircraft Rotation Schedule, which is different each week. This is due to the fact that each day many adaptations are made so as to minimise delays. For instance, if KLM know that an aircraft will arrive at Schiphol Airport with a delay, they could assign its next flight to

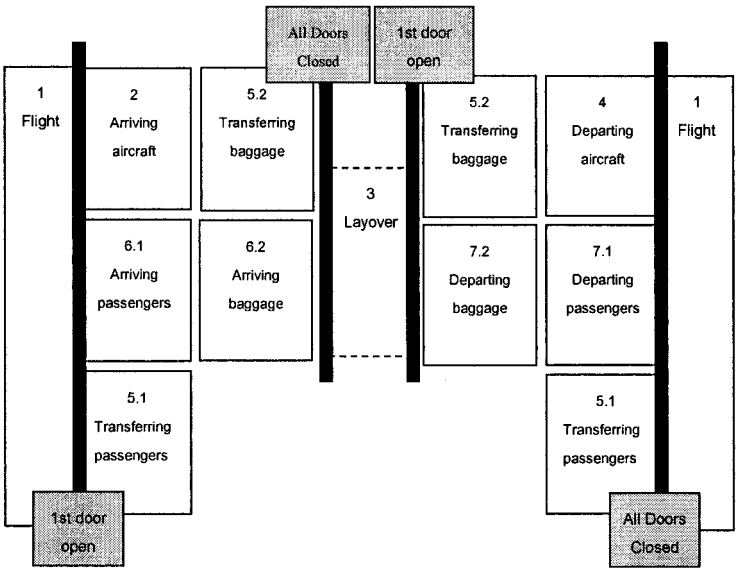


Figure 1. Figure 1. Building blocks sequence and relationships.

another aircraft so that that flight can still leave on time. Usually, KLM will also need other adaptations to have all flights fit into the Rotation Schedule again. When a schedule is first published, KLM do not know the exact layout of the Rotation Schedule, so they publish a hypothetical "average" one instead.

Before a schedule is published, an estimation of the expected punctuality (that is the percentage of "on time" flights) is performed using a simple deterministic model. As this model lacks accuracy, a simulation model is currently being developed in order to enable a better forecast. This model simulates aircraft movements according to a given schedule. The model subjects the schedule to a "stress test" by generating various disruptions such as air traffic congestion, delays during the boarding process or unexpected problems during maintenance. Throughout the simulation, a *Problem Solver* algorithm attempts to resolve delays by swapping flights in the Rotation Schedule, or in extreme cases, by cancelling flights. More successful runs of the simulation are considered as better schedules for implementation.

A simulation, though, has several disadvantages. Processing times are usually too long, which limits the number of schedules that can be assessed Also, KLM need to collect a huge amount of data about the processes that are being simulated. For the simulation model currently under development they need statistics about the variation in the actual flight duration, the variation in the

time it takes to handle an aircraft on the ground (boarding, fuelling, catering, etc), breakdown times of each aircraft type, etc. Each of these statistics must constantly be updated to reflect the change in flight routes, working methods, fleet, etc.

KLM are currently seeking a simpler model that would enable them to make a comparative statement, such as "Of a number of alternative schedules, schedule X will provide the best performance."

3. MODELS FOR THE PROBLEM

It was anticipated that there should be some features of any schedule that would be correlated with its performance. The first question is then what features should be investigated? A brainstorming session with representatives of KLM led to some suggestions. It was expected that the number of potential swaps available to a delayed flight would be an important factor, but measuring this value was not easy. In practice, it might also be necessary to undertake a cascade of swaps, so another possible measure of performance would be the length of time and/or the number of swaps needed to restore the schedule to its normal condition. However, this is also complicated to determine, although the *Problem Solver* module of the simulation could be invoked if necessary.

After further discussion, it was agreed to look at a simpler measure, which could easily be found, and is arguably a surrogate for some of the more complex measures suggested. This is the "Aircraft on Ground" (ACOG) measure which gives an indication of the number of aircraft on ground. ACOG can be calculated from the number of arriving aircraft, the number of layover aircraft, and the number of departing aircraft. Having obtained some features related to this measure, the next step is the identification of a suitable model for purposes of prediction. Candidates here include multiple linear regression methods, regression trees, neural nets and other pattern recognition techniques. However, the fact that the amount of data available was small meant that data-hungry methods should be avoided if at all possible. Thus it was resolved to begin the investigation with traditional statistical methods.

4. EXPERIMENTAL RESULTS

Eleven schedules were available (Summer/Winter 2000–02, apart from the last 13 weeks of 2002). KLM's operation at Schiphol is such that the activity occurs in four major waves—a deliberate strategy to maximise passengers' opportunities for making onward connections. Graphing the number of aircraft available on the ground reveals this pattern clearly. These can be counted in two ways: the more accurate picture is obtained by subtracting the lengths of BB2 and BB4, leaving just those aircraft that are actually idle at a given moment.

Table 1. Performance indicators for departure punctuality and arrival punctuality using two different models. For the predictor sets: p1, 1st peak; m, first moment (mean); sd, second moment (standard deviation); sk, third moment (skewness); k, fourth moment (kurtosis).

	Using BB3 only	Using BB2–4
PI Departures		
Predictor sets	p4m, p1sd, p1sk, p1k	p2m, p4m, p2sd, p4sd, p3sk
R-squared	95.6%	91.6%
p-value (F-test)	0.00032	0.01028
PI Arrivals		
Predictor sets	p4m, p1sk, p3sk, p3k	p1m, p4m
R-squared	95.2%	84.1%
p-value (F-test)	0.00042	0.00064

However, it is a simpler calculation to count the whole of the time on the ground from "First Door Open" to "Last Door Closed", which comprises the whole of BBs 2,3 and 4.

In the case of European operations, each day is more or less identical, so peaks can be defined quite easily. For each peak, the first four moments of the ACOG values were calculated for each day, using both definitions—BB3 and BB234. As days are so alike (apart from the very first day of a new schedule), one day can be selected at random as a representative of a schedule. As there are four peaks daily, we have 16 features as inputs, which we need to associate with the performance indicators (PIs) already calculated by KLM. The ones used for the models developed here were simply the departure and arrival punctualities: the fraction of planes (of those scheduled) that departed or arrived on time.

As a first step, correlations were calculated between the PIs and the 16 input variables. The six or seven most highly correlated input variables were than used in a stepwise regression procedure (using S-plus) to determine the best balance between parsimony and explanatory power (S-plus uses the Akaike information criterion for this purpose.) Table 1 summarises the models determined by this approach.

Of interest is the fact that p4m, the mean number of ACOG, is important for all four models, but the other predictors seem to be far less important. From KLM's point of view, this does not matter if the predictions are good enough, but from a modeller's perspective we would like to see more consistency. However, all models are based on just 11 data points, so perhaps the lack of consistency is not surprising. Prediction intervals can easily be obtained on the assumption of Normally distributed errors: these vary from $\pm 2\%$ for punctualities in the middle of the range to $\pm 3\%$ at the edges.

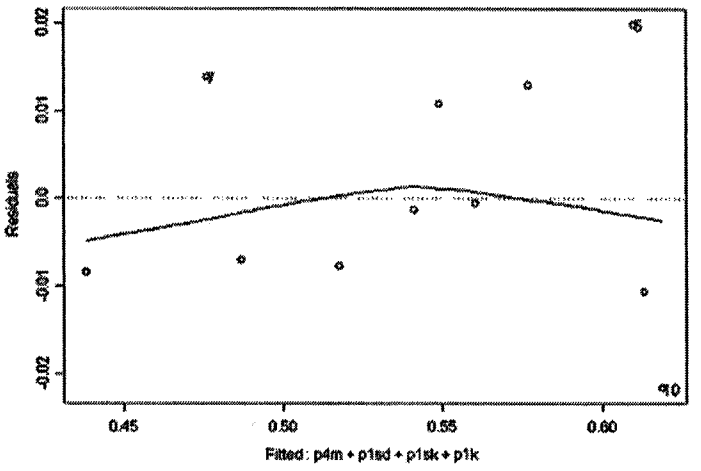


Figure 2. Residuals against fitted values for departure punctuality using BB3 only.

It was surprising that the R-squared values were as high as they were—we were anticipating that a linear model would be too simple, yet it seems quite powerful. Of course regression analysis makes certain assumptions about the errors, and it is necessary to check the residuals to see if these assumptions are plausible. The plot of residuals against fitted values was obtained for each model; in no case does a systematic pattern seem plausible, and a random scatter is obtained, as shown in Figure 2.

The three most extreme outliers (points 5, 7 and 10) are labelled; point 5 might well have been affected by September 11, but possible reasons for the others are not known. A smooth has been applied, but its slopes are not very steep, so the assumption that the errors are independent random variables seems plausible. Similar graphs were obtained for the other three models.

QQ plots of the residuals against Normal quantiles were also obtained. Figure 3 below shows the same case as in Figure 2.

The tails of the distribution in particular are not well fitted, so the assumption that the errors are Normally distributed is perhaps questionable. Thus any confidence intervals should be treated cautiously. In any case, the response variable in all four models is actually a ratio that is confined to remain between 0 and 1. This means that a better theoretical model would be based on a logistic transformation, since it is theoretically possible that a simple linear model could generate predictions outside the possible range of values. For example, we can hardly have a punctuality of greater than 100%! Such a model would also be based on a more plausible probability model than the Normal distribution.

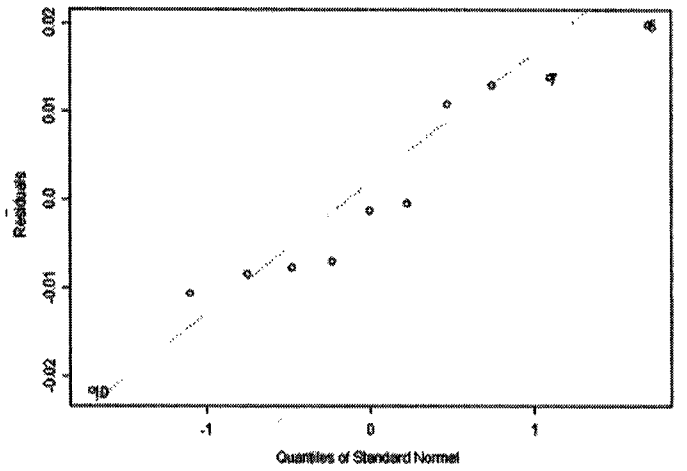


Figure 3. Normal QQ plot for residuals for departure punctuality using BB3 only.

However, attempts to fit such a model did not produce an improvement. A possible explanation is that the data available are all in the region of approximate linearity of the logistic curve. Consequently, any attempt to identify the turning points of the curve is likely to be rather speculative. In any case, on inspecting the coefficients of the models, it seems unlikely that we would predict bizarre fractions in practice. For example, using the most extreme values observed in the first model above would predict only 80% departure punctuality, and in the opinion of KLM's experts it is hard to imagine physical circumstances in which these values could be exceeded simultaneously (there is just not enough space to put many more planes, for example).

Thus, despite the attractions of a more plausible theoretical model, the airline is comfortable with the predictive ability of a simpler linear model.

5. CONCLUSIONS

An analysis of the expected number of aircraft on the ground has been shown to provide a good prediction for the robustness of a given schedule. Further refinements are possible—and desirable—but even this work has given the KLM's operations department a better insight into what makes a fleet schedule easier or harder to implement effectively. Some of the work that still needs to be done includes an analysis of the effect of day-to-day variations in the schedule—these variations are small, but preliminary work has suggested that the definition of activity peaks needs to be tighter, and the possibility of a day-of-the-week effect should also be explored. Furthermore, the schedules

examined so far have concentrated only on the European operations, where fleet homogeneity is substantial and diurnal variation is small. Incorporating the effects of the inter-continental timetable may lead to some changes in these conclusions.

Acknowledgments

We would like to thank the following bodies for supporting the collaboration described in this paper:

- The Lorentz Centre in Leiden: 45th European Study Group for Mathematics with Industry.
- The EU Mathematics, Computing and Simulation for Industry Network (MACSI-net).
- The Smith Institute for Industrial Mathematics and Systems Engineering.

References

Abara, J. (1989), Applying integer linear programming to the fleet assignment problem. *Interfaces*, **19**:20–28.

Argüello, M. F. and Bard, J. F. (1997), A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, **1**:211–228.

Barnhart C., Boland N. L., Clarke L. W., Johnson E. L., Nemhauser G. L. and Shenoi, R. G. (1998), Flight string models for aircraft fleeting and routing, *Transportation Science*, Special Issue on Airline Optimization, **32**:208–220.

Barnhart C., Lu, F. and Shenoi, R. (1997), Integrated airline scheduling. In *Operations Research in the Air Industry*, Gang Yu (Ed.), International Series in Operations Research and Management Science, Vol. 9, Kluwer, Dordrecht, pp. 384–403.

Barnhart, C. and Talluri, K. (1997), Airline operations research. In *Design and Operations of Civil and Environmental Engineering Systems*, A. McGarity and C. ReVelle (Eds.), Wiley, New York, pp. 435–469.

Ciesielski V. and Scerri P. (1997), An anytime algorithm for scheduling of aircraft landing times using genetic algorithms, *Australian Journal of Intelligent Information Processing Systems*, **4**:206–213.

Ciesielski V. and Scerri P. (1998), Real time genetic scheduling of aircraft landing times. In *Proceedings of the 1998 IEEE International conference on Evolutionary Computation (ICEC98)*, IEEE, Piscataway, NJ, pp. 360–364.

Clarke L. W., Hane C. A., Johnson E. L. and Namhauser G. L. (2001), Maintenance and crew considerations in fleet assignment, *Transportation Science*, **3**:249–260.

El Moudani, W. and Mora-Camino, F. (2000) A dynamic approach for aircraft assignment and maintenance scheduling by airlines, *Journal of Air Transport Management*, **6**:233–237.

Etschmaier, M. and Mathaisel, D. (1985) Airline scheduling: an overview, *Transportation Science*, **19**:127–138.

Feo, T. A. and Bard, J. F. (1989) Flight scheduling and maintenance base planning, *Management Science*, **35**:1415–1432.

Filar, J. A., Manyem, P. and White, K. (2001) How airlines and airports recover from schedule perturbations: a survey, *Annals of Operations Research*, **108**:315–333.

Gopalan, R. and Talluri, K. T. (1998) Mathematical models in airline schedule planning: a survey, *Annals of Operations Research*, **76**:155–185.

Hane, C. A., Barnhart, C., Johnson, E. L., Marsten, R. E., Nemhauser, G. L. and Sigismondi, G. (1995) The fleet assignment problem: solving a large-scale integer program, *Mathematical Programming*, **70**:211–232.

Hansen, J. V. (2004) Genetic search methods in air traffic control, *Computers and Operations Research*, **31**:445–459.

Jarrah, A. I. (2000) An efficient airline re-fleeting model for the incremental modification of planned fleet assignments, *Transportation Science*, **34**:349–363.

Klincewicz, J. G., Rosenwein, M. B. (1995) The airline exception scheduling problem, *Transportation Science*, **29**:4–16.

Love, M., Sorensen, K. R., Larsen, J. and Clausen, J. (2002) Disruption management for an airline—rescheduling of aircraft, *Applications of Evolutionary Computation: Proceedings of the EvoWorkshops 2002*, Lecture Notes in Computer Science, Vol. 2279, Springer, Berlin, pp. 315–324.

Mashford, J. S, Marksjo, B. S. (2001) Airline base schedule optimisation by flight network annealing, *Annals of Operations Research*, **108**:293–313.

Mathaisel, D. F. X. (1997) Decision support for airline schedule planning, *Journal of Combinatorial Optimization*, **1**:251–275.

Oussedik, S., Delahaye, D. and Schoenauer, M. (2000) Flights alternative routes generator by genetic algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, IEEE, Piscataway, NJ, pp. 896–901.

Rexing, B., Barnhart, C, Kniker, T., Jarrah, A. and Krishnamurthy, N. (2000) Airline fleet assignment with time windows, *Transportation Science*, **34**:1–20.

Richter, H. (1989) Thirty years of airline operations research, *Interfaces*, **19**:3–9.

Rosenberger, J. M., Schaefer, A. J., Goldsman, D., Johnson, E. L., Kleywegt, A. J. and Nemhauser, G. L. (2003) A stochastic model of airline operations, *Transportation Science*, **36**:4.

Ruland, K. S. (1999) A model for aeromedical routing and scheduling, *International Transactions in Operational Research*, **6**:57–73.

Rushmeier, R. A. and Kontogiorgis, S. A. (1997) Advances in the optimization of airline fleet assignment, *Transportation Science*, **31**:159–169.

Sosnowska, D. and Rolim, J. (2001) Fleet scheduling optimization: a simulated annealing approach. In *Practice and Theory of Automated Timetabling III (PATAT 2000)*, E. Burke and W. Erben (Eds.), Lecture Notes in Computer Science, Vol. 2079, Springer, Berlin, pp. 227–241.

Subramanian, R. (1994) Coldstart: fleet assignment at Delta Air Lines, *Interfaces*, **24**:104–120.

Talluri, K. T. (1996) Swapping applications in a daily airline fleet assignment, *Transportation Science*, **30**:237–248.

Thengvall, B. G., Yu, G. and Bard, J. F. (2001) Multiple fleet aircraft schedule recovery following hub closures, *Transportation Research* A, **35**:289–308.

Thengvall, B. G., Bard, J. F. and Yu, G. (2004) A bundle algorithm approach for the aircraft schedule recovery problem during hub closures, *Transportation Science*, to appear.

Yan, S. and Lin, C. G. (1997) Airline scheduling for the temporary closure of airports, *Transportation Science*, **31**:72–82.

Author Index