

Edmund K. Burke  
Hana Rudová (Eds.)

LNCS 3867

# Practice and Theory of Automated Timetabling VI

6th International Conference, PATAT 2006  
Brno, Czech Republic, August/September 2006  
Revised Selected Papers



 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Massachusetts Institute of Technology, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Moshe Y. Vardi

*Rice University, Houston, TX, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Edmund K. Burke Hana Rudová (Eds.)

# Practice and Theory of Automated Timetabling VI

6th International Conference, PATAT 2006  
Brno, Czech Republic, August 30–September 1, 2006  
Revised Selected Papers

## Volume Editors

Edmund K. Burke  
University of Nottingham  
School of Computer Science  
Jubilee Campus, Nottingham NG8 2BB, UK  
E-mail: ekb@cs.nott.ac.uk

Hana Rudová  
Masaryk University  
Faculty of Informatics  
Botanická 68a, Brno 602 00, Czech Republic  
E-mail: hanka@fi.muni.cz

Library of Congress Control Number: 2007941262

CR Subject Classification (1998): F.2.2, G.1.6, G.2, I.2.8

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743  
ISBN-10 3-540-77344-4 Springer Berlin Heidelberg New York  
ISBN-13 978-3-540-77344-3 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

[springer.com](http://springer.com)

© Springer-Verlag Berlin Heidelberg 2007  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper SPIN: 12205875 06/3180 5 4 3 2 1 0

# Preface

This volume contains a selection of the papers presented at the Sixth International Conference on the Practice and Theory of Automated Timetabling (PATAT) which was organized in Brno, Czech Republic, from August 30 to September 1 of 2006.

The PATAT conferences, which are held every 2 years, bring together researchers and practitioners from across the broad spectrum of inter-disciplinary research activity in search methodologies for automated timetable generation. This includes university timetabling, school timetabling, personnel rostering, transportation timetabling, sports scheduling. The programme of the 2006 conference featured 70 presentations which represented the state of the art in automated timetabling: there were four plenary papers, 17 full papers, 41 extended abstracts, and eight system demonstrations. After the conference, all authors were invited to submit their papers to a second round of rigorous refereeing for this volume of selected revised papers. We are pleased to have accepted 25 papers for this volume. This figure represents the highest number of acceptances in a PATAT post-proceedings volume and is a testament to the high standards of the papers that were submitted.

The organization of the book is structured around particular problem areas. Several papers are devoted to employee timetabling. It is not surprising to see that most of these papers are concerned with health care personnel scheduling which, historically, has always been an active application area at PATAT. The last, more general, paper in this section studies the relationship with production scheduling. The section on sports timetabling contains papers which are mostly oriented towards various tournament timetabling problems. The last article on referee assignment has a close relationship with employee timetabling. As usual, the volume contains a wide range of papers on educational timetabling. During the conference, various discussions were generated by Barry McCollum's plenary talk where questions on the applicability of the current academic research on real-world university timetabling were posed. This is an important issue and it is likely to impact significantly upon future timetabling research.

Of course, a very important aspect of the presented papers, across all applications, is represented by the search methodologies that are presented and discussed. This volume presents a wide variety of new and innovative techniques which represent an important contribution to the timetabling research literature. We think that this work provides a strong platform for the future and we look forward to the ongoing success of the conference series.

The meeting in Brno was the sixth in the PATAT series of international conferences. The first five conferences were held in Edinburgh (1995), Toronto (1997), Konstanz (2000), Gent (2002), and Pittsburgh (2004). Selected papers

from these conferences have all appeared in the Springer series. The full references are:

Edmund K. Burke and Peter Ross (Eds.): Practice and Theory of Automated Timetabling, 1st International Conference, Edinburgh, UK, August/September 1995, Selected Papers, Lecture Notes in Computer Science, Vol. 1153, Springer, 1996.

Edmund K. Burke and Michael Carter (Eds.): Practice and Theory of Automated Timetabling, 2nd International Conference, Toronto, Canada, September 1997, Selected Papers, Lecture Notes in Computer Science, Vol. 1408, Springer, 1998.

Edmund K. Burke and Wilhelm Erben (Eds.): Practice and Theory of Automated Timetabling, 3rd International Conference, Konstanz, Germany, August 2000, Selected Papers, Lecture Notes in Computer Science, Vol. 2079, Springer, 2001.

Edmund K. Burke and Patrick De Causmaecker (Eds.): Practice and Theory of Automated Timetabling, 4th International Conference, Gent, Belgium, August 2002, Selected Papers, Lecture Notes in Computer Science, Vol. 2740, Springer, 2003.

Edmund K. Burke and Michael Trick (Eds.): Practice and Theory of Automated Timetabling, 5th International Conference, Pittsburgh, USA, August 2004, Selected Papers, Lecture Notes in Computer Science, Vol. 3616, Springer, 2005.

Edmund K. Burke and Hana Rudová (Eds.): Practice and Theory of Automated Timetabling, 6th International Conference, Brno, Czech Republic, August–September 2006, Selected Papers, Lecture Notes in Computer Science, Vol. 3867, Springer,. (This volume.)

The seventh conference will be held in Montreal, Canada, August 2008. See <http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml> for information on the conference series.

We would like to express our gratitude to the large number of people who contributed to the excellent conference in Brno and who worked very hard to prepare this volume. The Steering Committee ensures that the series goes from strength to strength. The Programme Committee represents the pool of referees for both rounds of the reviewing process. The papers for this volume were carefully refereed by four members of the Programme Committee. Their insight has meant that many of the papers were significantly improved during the reviewing process. We should also extend our thanks to all the authors who carried out a significant amount of work to produce papers of the current high level of quality. Our thanks also go to Piers Maddox, our copy editor, who, as usual, prepared the volume to an extremely high standard of formatting and typesetting.

We would particularly like to thank the Faculty of Informatics at Masaryk University for hosting the conference. We are also grateful to all the members

of the Organizing Committee, who helped so much to ensure that the conference was a success. A very special thank you should go to Adam Rambousek for his support and for granting us the permission to use his conference management system. We would also like to express our gratitude to Jakub Mareček for his assistance with the typesetting of the conference proceedings and Lenka Bartošková for her assistance with the budget. Particular thanks should also go to Emma-Jayne Dann for her administrative support. Last but not least, we would like to thank the conference sponsors: ORTEC bv, eventMAP Ltd, CEL-CAT, AVmedia, a.s., and the Ministry of Education, Youth and Sports of the Czech Republic for their support under research intent No. 0021622419.

We are, of course, also very grateful to all the delegates. They helped to create a wonderful atmosphere at the conference in Brno. We are looking forward to seeing them together again at the next conference in Montreal in the Summer of 2008.

July 2007

Edmund K. Burke  
Hana Rudová

# Organization

## Programme Committee

Edmund Burke (Co-chair)	University of Nottingham, UK
Hana Rudová (Co-chair)	Masaryk University, The Czech Republic
Hesham Alfares	King Fahd University, Saudi Arabia
Viktor Bardadym	Noveon Inc., Belgium
James Bean	University of Michigan, USA
Peter Brucker	University of Osnabrück, Germany
Michael Carter	University of Toronto, Canada
Peter Cowling	University of Bradford, UK
Patrick De Causmaecker	Katholieke Universiteit, Leuven, Belgium
Kathryn Dowsland	Gower Optimal Algorithms Ltd., UK
Andreas Drexl	University of Kiel, Germany
Wilhelm Erben	University of Applied Sciences Konstanz, Germany
Jacques A. Ferland	University of Montreal, Canada
Michel Gendreau	Centre de Recherche sur les Transports, Montréal, Canada
Alain Hertz	Ecole Polytechnique de Montréal, Canada
Jeffrey H. Kingston	University of Sydney, Australia
Raymond Kwan	University of Leeds, UK
Gilbert Laporte	Université de Montréal, Canada
Vahid Lotfi	University of Michigan-Flint, USA
Barry McCollum	Queen's University and eventMAP Ltd., UK
Amnon Meisels	Ben-Gurion University, Beer-Sheva, Israel
Keith Murray	Purdue University, USA
Thiruthlall Nepal	Durban Institute of Technology, South Africa
Ender Özcan	Yeditepe University, Turkey
Ben Paechter	Napier University, Edinburgh, UK
Gilles Pesant	Ecole Polytechnique de Montréal, Canada
Sanja Petrovic	University of Nottingham, UK
Jean-Yves Potvin	Université de Montréal, Canada
Celso Ribeiro	Universidade Federal Fluminense, Brazil
Rong Qu	University of Nottingham, UK
Andrea Schaerf	Università di Udine, Italy
Jan Schreuder	University of Twente, Enschede, The Netherlands
Jonathan Thompson	Cardiff University, UK
Paolo Toth	University of Bologna, Italy
Michael Trick	Carnegie Mellon University, USA



Greet Vanden Berghe	KaHo St-Lieven, Belgium
Stefan Voss	University of Hamburg, Germany
Dominique de Werra	EPF-Lausanne, Switzerland
George M. White	University of Ottawa, Canada
Michael Wright	Lancaster University, UK
Jay Yellen	Rollins College, USA

## Organizing Committee

Hana Rudová	Chair
Tomáš Černý	Poster and Web Design
Dagmar Janoušková	
Dalibor Klusáček	
Iva Krejčí	
Petra Křivánková	
Jakub Mareček	ISBN Proceedings
Adam Rambousek	Conference Management System
Martin Šmerek	

Emma-Jayne Dann

## Steering Committee

Edmund K. Burke (Chair)	University of Nottingham, UK
Ben Paechter (Treasurer)	Napier University, Edinburgh, UK
Patrick De Causmaecker	Katholieke Universiteit, Leuven, Belgium
Wilhelm Erben	University of Applied Sciences Konstanz, Germany
Jeffrey H. Kingston	University of Sydney, Australia
Amnon Meisels	Ben-Gurion University, Beer-Sheva, Israel
Hana Rudová	Masaryk University, The Czech Republic
Michael Trick	Carnegie Mellon University, USA
George M. White	University of Ottawa, Canada

# Table of Contents

## General Issues

A Perspective on Bridging the Gap Between Theory and Practice in University Timetabling .....	3
---	---

Very Large-Scale Neighborhood Search Techniques in Timetabling Problems .....	24
---	----

Measurability and Reproducibility in University Timetabling Research: Discussion and Proposals .....	40
--	----

## Employee Timetabling

Physician Scheduling in Emergency Rooms .....	53
---	----

A Flexible Model and a Hybrid Exact Method for Integrated Employee Timetabling and Production Scheduling .....	67
--	----

Memes, Self-generation and Nurse Rostering .....	85
--	----

An Evaluation of Certain Heuristic Optimization Algorithms in Scheduling Medical Doctors and Medical Students .....	105
---	-----

## Timetabling of Meetings

Scheduling Research Grant Proposal Evaluation Meetings and the Range Colouring Problem .....	119
--	-----

## Sports Timetabling

Constructive Algorithms for the Constant Distance Traveling Tournament Problem .....	135
--	-----

Scheduling the Brazilian Soccer Tournament with Fairness and Broadcast Objectives ..... 147

Referee Assignment in Sports Leagues ..... 158

A Branch-and-Cut Algorithm for Scheduling the Highly-Constrained Chilean Soccer Tournament ..... 174

**Course Timetabling**

Modeling and Solution of a Complex University Course Timetabling Problem ..... 189

Timetabling Problems at the TU Eindhoven ..... 210

The Teaching Space Allocation Problem with Splitting ..... 228

Solving the University Timetabling Problem with Optimized Enrollment of Students by a Self-adaptive Genetic Algorithm ..... 248

**School Timetabling**

A Case Study for Timetabling in a Dutch Secondary School ..... 267

Scheduling School Meetings ..... 280

Hierarchical Timetable Construction ..... 294

The KTS High School Timetabling System ..... 308

**Examination Timetabling**

A Novel Fuzzy Approach to Evaluate the Quality of Examination Timetabling ..... 327

Linear Linkage Encoding in Grouping Problems: Applications on Graph Coloring and Timetabling .....	347
Ant Algorithms for the Exam Timetabling Problem .....	364
An Extensible Modelling Framework for Timetabling Problems .....	383
An Experimental Study on Hyper-heuristics and Exam Timetabling ....	394
<b>Author Index</b> .....	413

# **General Issues**

# A Perspective on Bridging the Gap Between Theory and Practice in University Timetabling

Barry McCollum

School of Electronics, Electrical Engineering and Computer Science,  
Queen's University, University Road, Belfast BT7 1NNN, Ireland  
b.mccollum@qub.ac.uk

**Abstract.** The study of the relationship and interaction between the work carried out in the academic literature and the requirements of university administrators is essential if ideas generated by research are to benefit every-day users. Conversely, it is crucial that the needs of the timetabling community influence the direction taken by research if high-quality practical solutions are to be produced. A main objective of the work presented here is to provide up-to-date information which will enable researchers to further investigate the area of timetabling research in relation to the generation of robust and flexible techniques which can cope with complexities experienced during implementation in 'real world' scenarios. Furthermore, although not discussed here in detail, it is essential, from a commercial perspective, that these developed leading edge techniques are incorporated and used within general applicable timetabling tools. The aim of this paper is to motivate the discussion required to *bridge this timetabling gap* by bringing timetabling research and educational requirements closer together.

## 1 Introduction and Context

In the recent international review of Operational Research in the UK (commissioned by the Engineering and Physical Sciences Research Council), a major identified weakness in the current approach to Operational Research is described as follows [50]:

... a gap still remains between the output of a successful research project and what is needed for direct use by industry.

In general, the area of educational timetabling is one such area. Our research-based spin-out company, eventMAP Limited, has an important role to play with respect to this 'gap' as it is in a unique position to integrate leading edge research techniques with the requirements of the user base in the provision of timetabling solutions. One of the primary overall aims of the company is to specify software which acts as an enterprise resource planning tool as well as a management information service, informing on strategic ways forward for the need for, use of and allocation of resources within an institution. A major aspect of the adopted strategy for achieving this is to highlight the important aspects of institutional

requirements to researchers in the field while updating algorithmic techniques within the software, thus enabling research solutions to be produced which are both workable and of a high quality. The intention of this paper is to focus on the initial part of the strategy by reporting on the needs of educational institutions from a practical point of view in terms of two main areas where timetabling is required, i.e. examination and course timetabling. In each area, a number of challenges are detailed which are based on the author's experience of working in the area from both an academic and practical view point. It is stressed that these challenges certainly do not represent all of the issues that require work from researchers, rather they represent a selection of key themes which, in the author's view, will help bridge the identified gap and move the area of educational timetabling to a new level both in research and practical terms.

## 2 Examination Timetabling

The examination timetabling problem, studied in numerous papers in the PATAT conference series [15,16,22,40,61], is characterized by a set of students taking a set of exams over a specified time period within the context of various constraints. The quality of the timetable is normally measured as a function of best spread of examinations per student though some notable exceptions do occur [8,70]. Various algorithms have been used with their effectiveness being measured in relation to a standard set of benchmark data. An up-to-date review is provided in [72]. In addition to the PATAT Conference series, many papers have been published on specific techniques along with reporting of various surveys [37,76]. It is worth noting that research in this area has been instrumental in the continued development of the field of search methodologies and, in particular, metaheuristics. Although it is not intended to provide a general commentary on the approaches adopted to date it is possible to argue that the nature of the gap between research and practice has not been aided by the simplicity of the current datasets, e.g. the lack of substantial benchmark data with sufficient room, constraint and solution modelling data. It is expected that the release of six new datasets [30] along with a dedicated web service to the research community via the web site at <http://www.cs.nott.ac.uk/~rxq/data.htm> will go a long way to remedying this situation. This service will also act as a repository of information relating to techniques and solutions generated and will enable researchers to easily and accurately test and compare approaches.

EventMAP Limited released the latest version of its flagship examination product, Optime<sub>xam</sub>, in January of 2006. An earlier version of the software was presented at the PATAT conference in Konstanz, 2000 [61]. The additional functionality made available through this new version will be discussed at the conference during a software presentation [23]. In general, the aim of improving Optime<sub>xam</sub> is to make the system as intelligent and intuitive as possible, providing maximum information to the institutional administrator, allowing informed strategic and managerial decisions to be made. This has been achieved through the inclusion of the user in all stages of the 'examination modelling' process. It

is important to note that although not described in detail here, the ‘gap’ between the needs of the user and the provision of software is also being tackled within the company by the development of a close working relationship with users. Feedback from this process which is relevant to researchers includes modelling aspects of the information, algorithmic and solution development, all of which represent significant challenges for the research community. The following discussion is concentrated around this reported examination modelling process.

## 2.1 Building the Institutional Model

The development of examination timetables within institutions is a multi-phase procedure that is dependent on varying criteria at each stage. Firstly, a structure has to be decided on before exams and students are assigned, e.g. the length and format of the time period together with the ‘diet’ of rooms which are to be made available. Secondly, data on exams and associated constraints have to be added before the student information is considered. The stage and degree of automation is highly dependent on the procedures adopted within the institution. This multi-stage process is referred to here as building the ‘institutional model’. This process encompasses two main aspects: information and solution modelling.

**Information Modelling.** Information modelling can be divided into data and constraint modelling. The base examination data from which a workable solution is achieved are composed of student enrolment, exam and space data. In addition, the construction of an overall solution is phased due to the information environment within which the examination process takes place. In practice, a solution is often attained based on a percentage of the actual data due to incomplete and inaccurate data from the student administration systems. Ultimately the algorithms applied must therefore construct solutions working with a degree of uncertainty. The inadequacies of the data set-up therefore represent the first challenge to the timetabling community. It is suggested that there are two possible approaches to solving this problem, i.e. either solutions are sought with associated repair mechanisms or robust optimisation techniques are used which produce solutions that are ‘good’ for an agreed range of input values. Under this scenario, a solution would be sought that remains feasible for all potential input data values. Although some work is evident in the literature in relation to the first of these approaches in relation to educational timetabling [44,54], little attention has been paid to the second.

Constraint modelling involves setting up a range of criteria which effectively describes the boundaries within which a solution should be constructed. Constraints used in institutions have been reported in 1996 [21]. Since then, in the UK in particular, there has been a steady increase in complexity regarding this issue with the implementation of increasingly flexible modular course structures by many universities. The central production and coordination of the associated examination timetable has become increasingly difficult with more examination offerings having to be timetabled in such a manner so as to offer students maximum spread throughout the session while ensuring space usage is maximised. In



addition, many new constraints have been added to the overall problem to accommodate all types of special needs of students. An example of this was reported in the Times Higher Educational Supplement in March of 2006 where students from a Muslim background require Fridays free of examinations [79]. This and other additional soft constraints further complicate the modelling process and the scope of potential solutions. It is essential these are documented and incorporated into the modelling process as, for example, at our leading implementation site, 9% of students in the 2004/05 academic year had special needs with regards to their examination requirements. The second challenge is therefore to redefine the problem in terms of recent identified changes. This can be achieved by getting access and reporting on practical examples of constraints and the processes involved. The PATAT conference series and the close link with eventMAP limited is of particular relevance here as practical issues as well as datasets can be added to the research base on a continual basis. Another important aspect of constraint modelling is the structure of the examination session, i.e. session modelling. Two features of this are detailed below.

In establishing an institutional model for the examination process, one of the major issues for many institutions is the potential relaxing of a constraint which has hitherto been considered ‘hard’, i.e. the imposing of certain time periods within the day structure. For example, a day may be split into two periods of three hours in length, one beginning at 9AM and the other beginning at 2PM. Analysis of various solutions produced by eventMAP has shown that this is the single biggest factor in relation to poor usage of time and space and hence a major contributory factor to poor overall solutions. This is chosen here as it is an excellent example of a hard constraint which needs to be changed to move the examination timetabling forward from a practical point of view. Before leaving the established ‘period based’ approach to one side, it is essential to understand the required needs and the extent of ‘non-period’ based timetabling. The period based nature of the problem needs to be investigated to establish a model where examinations can be scheduled during any part of the defined day. This issue is related to recent work with respect to a redefinition of the nurse scheduling problem [18] where metaheuristic techniques which have been used to manage this time interval coverage have produced the best results so far on the presented data. Due to the similarity of the nurse rostering and examination timetabling problems it is considered appropriate that these techniques are investigated. The concept of ‘time interval’ was introduced, where instead of formulating the staff requirements as the number of personnel needed per shift type for each day of the planning period, time interval requirements allowed for the representation of the personnel requirements per day in terms of start and end times of personnel attendance. As with the nurse scheduling example, an updated formulation would enable the provision of a greater number of time slots and would reduce the amount of unproductive time currently in existence.

It is clear that institutions involved in the process of carrying out the initial stage of the institutional modelling process often do so blindly. That is to say, they base the timetable on new data but attempt to superimpose this on

existing models of how the examination sessions should progress. For example, an existing model for a particular institution may be a certain number of periods over a designated time period with a certain number of rooms. This, in part at least, is related to inadequate methods which allow users to understand how solutions are being created. For example, space considerations are often an afterthought with the primary aim being the actual creation of a timetable. No help is afforded to the users in directing them towards a solution which is ‘right’ for the institution. Before going on to the important issue of solution modelling in the next section it is important to note that the investigation of similarity of data to previous datasets from the same or indeed other institutions is important if efficient and effective models are to be found. Continuing on from recent work [19,20] on similarity measurements between datasets, novel techniques need to be investigated to establish how changes in individual datasets from year to year affect the nature of the examination set up and ultimately the algorithmic methods applied.

**Solution Modelling.** Solution modelling is concerned with the construction of a solution in terms of what is deemed important to the institution. Currently, the majority of the work in evaluating a solution is based on the production of a single solution from each execution of the algorithm whose value is measured by a single objective weighted sum of soft constraints. There are some exceptions though: for example, in paper [8], the quality of a constructed timetable is considered in terms of the average penalty per student and the highest penalty imposed on any one student. Although research has been carried out in modelling the problem as a multi-criteria/objective problem [14,69] this work has not yet been implemented into a generalised tool. The responsibility is currently on the user to model the problem accurately at the constraint modelling phase and subsequently ‘leave’ it to the algorithm to produce the ‘best solution’. This has the effect of the user feeling ‘frozen’ out of the solution construction phase and gives the impression that this is the best solution based on the constraint set-up process. Of course, this is not the case with many solutions being possible which ‘best’ fit the constraints set-up. Paquete et al. [67] carried out work in which individual constraints were given preference at various stages of the process. This is similar to how the process of solution construction is carried out in a number of institutions with, for example, the effectiveness of a solution being measured as the ‘number of students with two examinations in a day’. It is clear that the user requires a number of solutions to be presented with the differences explained intuitively, thus allowing the user to decide on what solution is the ‘best’ to meet the institutions needs. It is suggested here that this could be achieved by a combination of techniques incorporating Pareto optimization and fuzzy techniques: e.g., the user chooses the characteristics of the solutions they would like to see from a number of fuzzy sets. This could possibly be translated into a choice function for discriminating between the non-dominated Pareto solutions generated by a multi-objective algorithmic technique. It is stressed that this is only one possible approach which could be used to address this important issue. More work is required on how the quality of solutions are measured. The

challenge for researchers is the provision of a solution where the user understands the trade-offs between the original objectives.

Once a solution is being generated, it is normal to have a construction phase followed by an improvement phase. In both cases there have been many heuristic techniques applied (see [37]). Recent work has shown promise in relation to using a combination of heuristics in relation to the initial construction [7]. Results on the benchmark datasets have got increasingly better over the years as more and more metaheuristic techniques have been applied and domain-specific knowledge has been increasingly incorporated into the approaches [72,37]. One criticism of this approach is that the developed techniques have become specialised in relation to the benchmark datasets at the possible cost of generality, i.e. techniques which can produce ‘good’ results when applied across a wide range of other real-world scenarios. Recently, in terms of metaheuristics, it has been shown that changing the neighbourhood structure has been effective. It is felt that the hyperheuristics approach (heuristics to choose heuristics) [31] undoubtedly offers promise as this methodology is based on raising the level of generality by aiming to automatically apply the correct heuristic or metaheuristic at the correct stage of the problem, be that in the construction or indeed the improvement phase. Currently, Optime enables the timetabling algorithm to be varied depending on the user algorithmic modelling process. These observations are the result of a close working relationship with five principal users in the UK and they currently represent the basis of further research [30]. Currently the combinations of algorithmic structures available are Saturation degree (Heuristic Method) [36], Adaptive [34] and Great Deluge during an additional improvement cycle [33]. The algorithm set-up thus enables the user to have control over the time spent on various aspects of its operation. This is a first step in involving the user at a higher level of the algorithmic modelling of the problem and is in response to the observation that various algorithmic set-ups perform better on different datasets. It is important to understand why various metaheuristic and combination of metaheuristics work better in particular situations. One challenge to the research community is therefore to explore how new search methodologies [11,2,13,34] can underpin the development of more widely applicable timetabling systems. Indeed this is one of the main motivating factors for the current level of interest in hyperheuristic research [25,26,31].

### 3 Course Timetabling

The university course scheduling problem is concerned with groups or classes of students following a particular defined pathway or course which has associated events that require the allocation of time and resources. Recent definitions of the course timetabling problem can be found in [74,76]. As with the university examination problem, a solution requires a number of hard and soft constraints to be satisfied. Similarly, the central production and coordination of the course timetable is essential as more modules and associated events have to be timetabled in such a manner as to, firstly, offer students maximum flexibility of choice,

secondly, to provide flexibility for staff and, thirdly, to ensure that teaching space is used effectively. Universities, struggling with rising student numbers, have increasingly relied upon the automation of this task to produce efficient timetables which satisfy these constraints [37]. Much of the software assistance that is currently available is either a commercial product or has been designed specifically for the institution in which it was developed [10,42,68]. In both cases the timetabling process often involves significant human interaction which, in practice, can turn the process into a room booking exercise [56,60]. Therefore, the construction of a solution is often categorised by finding any timetable that satisfies all of the constraints [76]. From a software point of view, any solution is often seen as a good solution and, indeed, the notion of an ‘optimised solution’ is usually not a main objective of incumbent university administrators. The reasons for this are diverse and complicated. One issue is that as too much assumed and incomplete knowledge surrounds the entire process and there exist many staff, with differing viewpoints involved. The data required for the process are often difficult to obtain and, as with the examination process, are often ‘sketchy’ [65,75]. From a staff point of view, fixed views exist on when and where teaching should take place within a predominantly ‘territorialist’ culture [56]. These issues will be further explored in the remainder of the paper with challenges presented as to how this area can be moved forward from a research point of view. It is important to note that, within the majority of universities which use automated systems, the process of the production of a workable timetable remains firmly with a combination of lecturing and administrative staff rather than the sole use of the automated component. Recent years have seen significant research efforts to improve this situation. The following papers represent a small selection of these contributions: [3,41,42,54,56,58,60,65,74]. Carter [41] stressed the importance of taking into consideration and dealing with the human factors associated with the process of constructing an institution-wide timetable. However, when dealing with the issue of course timetabling, it is often the case that many of the papers ignore the human factors all together, choosing to deal with ‘sculpted’ datasets in order to evaluate particular techniques and approaches. Some real-world aspects have been discussed in the literature but these tend to be in conference abstracts (as a small selection, see [45,46,51,66,75]) rather than full papers. If one of the strategic goals of timetabling research over the next few years is to close the gap between theory and practice then these issues have to gain more prominence in the mainstream literature.

Although many advances have been made with respect to the development of search techniques on benchmark datasets [3,4,64,74,77], there is not much evidence that the work has been translated into actual implementations within a significant number of institutions. Indeed Carter and Laporte [42] comment that they were ‘somewhat surprised to discover that there are very few course timetabling papers that actually report that the (research) methods have been implemented and used in an institution’. Although this was reported almost a decade ago, the situation largely remains unchanged. They go on to say that

they expected to see a number of implementations in the near future. Once again, unfortunately this has largely not been the case.

In relation to this area in general, it is suggested here that there has been insufficient investigation of real-world issues and therefore understanding of the methodologies used by expert timetablers. More work needs to be carried out on the formulation and modelling of the problem. This latter issue is particularly challenging because different institutions must satisfy a range of different constraints in generating an institution-wide timetable [58,42] which means that a generally applicable solution to this complex problem is extremely difficult. Given the complexities of real-world course scheduling, many researchers have developed approaches which rely on various simplifying assumptions in modelling the problem. While it can be argued that this is valid as an initial research test bed, which has resulted in useful and powerful search techniques, such an approach needs to be supplemented by methods which address the true complexities of the problem that must appear in real-world applications. By way of illustrating this point, recent work carried out on practical course timetabling by the Metaheuristic network [64] used generated datasets. It was stated that

The problem we are studying in the Metaheuristics project is one that is closely based on real-world problems, but simplified. We are not entirely happy about using a simplified problem, but the reasons are two-fold: We want to be able to see more clearly what is going on in algorithms designed to solve the problem. Real data is too complicated, and real problems have too many soft and hard constraints to allow researchers to properly study the processes...

The large number of soft and hard constraints in real data (and the differences between them at different institutions) make it a long process for researchers to write code to solve them, or to adapt existing programs to be suitable.

Although this has been useful, from a practical point of view, the results obtained do not seem relevant in practice. In addition, the impression is often that benchmark course timetabling datasets [64,77] are seen as data which can be used in addition to examination datasets to prove that certain search techniques are of benefit. Although successful in this regard the gap between research techniques and the software required for actual implementations is much wider than that seen with examination timetabling. Whereas this paper has spent the opening sections detailing challenges which will help narrow the gap in relation to examination timetabling, the rest of the paper will concentrate on describing course scheduling from a practical point of view with the hope of identifying what is required if a relevant and comprehensive formulation of the problem is to be reached. It is felt that this view of the course timetabling problem will better serve the purpose of making timetabling research more relevant to real-world practice. It is stressed that the contribution of timetabling research must address more wide-ranging issues than the tuning of algorithms to work well on particular datasets. Rather, the modelling issues related to the complexity of real-world implementations must be recognised and dealt with. The most

realistic formulation of the problem which currently exists can be found at [48]. Further work is required to build on this to allow the full complexities of the problem to be explored and to narrow the current gap. With this aim in mind, it is essential that more comprehensive representative benchmark datasets are made available along with information on the aims of the associated institution.

### 3.1 A Very Different Timetabling Problem

University course timetabling is often reported in the literature as a variant of the related examination timetabling problem [76]. Indeed it is the author's impression that many pieces of research default to talking about examination timetabling when they are talking about university timetabling in general. Although some of these issues are further described in subsequent sections of the paper it was felt worthwhile to draw out the major differences between the two types of timetabling at this early stage in the discussion. The reported difference is often the addition or removal of particular constraints: e.g., more than one event cannot take place in the same room, and lectures should be avoided in the last period of the day [3]. In addition, the term 'best spread' of events has an entirely different meaning.

A major difference with the examination timetabling process is the environment in which the construction process is carried out. This is a dynamic, multi-user distributed environment with various cohorts of schools and departments who often operate quite autonomously. Although issues in relation to this have been studied, for example [49,63,71,75], much more work is required on understanding the issues involved and the interplay between user interaction and managing the information with the goal of producing a workable solution and the extent to which techniques can be used in an automated process. These issues will be discussed further at various places under the heading of 'building the institutional model'.

Another difference that is often overlooked is that, as with the examination problem, course timetabling does not take place at the module or course level. The following presents a discussion on the effects of this. Consider the module 'Introduction to Computer Science' with associated module number 110CSC101. The associated examination for the module will normally take place at the end of the semester in which the module is given and will be timetabled by the rules employed by the institutional examination officer which are generally those governing the body of research which has taken place over the last decade or so. Therefore, in this case the 'gap' which exists between what is required by the institution and the techniques researched from an academic sense, is small. The course timetabling issues with the module 110CSC101 are more complicated. The module can be broken into a series of events which require timetabling: e.g., lectures, seminars, tutorials, practical classes and laboratory classes. A subset or indeed all of these 'event types' require timetabling in a manner which provide the group of students associated with the module, firstly, a feasible solution and secondly, a 'good' timetable. A feasible solution is achieved by ensuring that individual students can attend all event types associated with each of the

modules that constitute the overall pathway they are enrolled on: e.g., year one of BSc in Computer Science. Secondly, a 'good' solution is one which satisfies the soft constraints as defined by the institution: e.g., lectures should be in the morning in a particular time or room. It is clear that these soft constraints require a higher investigation as they can vary from one institution to another and indeed from one event type to another belonging to the same module. Furthermore, in setting up the problem, these events have different individual requirements, ordering and constraints. The following section outlines some of the associated issues.

The simplest example is that particular event types are usually associated with certain types of space: e.g., a computer laboratory class must take place in a computer laboratory. Also, lecture events represent the entire group of students on the module whereas the other event types represent subgroups as students are divided into smaller groups for different types of study. This issue of event subdivision is further explored in the following section. From an ordering perspective, it is often the case that particular orders of events over a defined time period, e.g. a week, are defined to achieve the desired combination of teaching and learning skills. It is also often the case that particular events are related to each other in relation to the time which separates them in this ordering: e.g., seminar classes should be timetabled in the afternoon following the lecture activity. In addition there is an associated hierarchy with the event types: e.g., lectures are timetabled as a priority in the first instance to ensure that the entire group can be brought together. It is often the case that this situation means that lectures will be timetabled first with all other events timetabled after week one of the semester. Of course, there are many variations of this related to when the timetable is produced in relation to student enrolment: i.e., pre-enrolment or indeed post-enrolment. Event types may also have a particular life span associated with them throughout the semester. Whereas the lecture event may run in a particular format throughout the entire semester, other event types may begin and end in particular weeks. In addition they may have an associated pattern which is individual to the event type: e.g., lectures may run twice a week for 12 weeks whereas lab classes may begin in week three and run for a three hour afternoon slot every two weeks for six weeks. Currently, research does not take these considerations into account when either defining the problem or applying techniques to help solve the problem. This has been detrimental to the overall practical area and has meant researchers, in many cases, have been working on oversimplified problems.

Course scheduling, much more than examination timetabling, must be seen in the wider context of the use and availability of institutional space either existing or in the planning stage. This linkage allows measured and improved utilisation while identifying the needs for particular types of space across the institution. The company eventMAP Limited aims to model how increases in course delivery, through effective timetabling, can affect the overall nature and structure of the campus. Ultimately, this would allow for strategic decisions to be taken in relation to room types, sizes and quantities across all space types

within the Institution. The course timetabling system is therefore a fundamental part of the strategic computing systems within the institution.

Another major difference with the examination timetabling problem is not only related to differences in the nature of the information and constraints but in the style in which the solution is constructed. Overwhelmingly in all consultancy and implementation undertaken to date within eventMAP Limited, the timetable is constructed prior to student enrolment and therefore optimised on projected student numbers taking particular combinations of modules. In many cases the goal of optimisation is sacrificed for the sake of getting a solution which is workable. Student clashing is related to defined course structures as opposed to the examination counterpart which is based purely on student enrolment to assessment events. Regarding soft constraints, the emphasis is on the ability to offer as many options as possible as opposed to best spread across a particular examination session. Administrators employ heuristics that suggest what modules should be made available to particular courses and which ones should not. Indeed, this information can often be inferred from the previous year's data or obtained directly from members of particular schools. Because the timetable is constructed pre-enrolment, inefficiencies occur which are allowed to ripple throughout the rest of the year. After the initial construction, potentially the solution could be reshuffled or indeed amended based on a different measure of optimisation. This option is not presently favoured by institutions due to the disruption that would be caused. There are a number of reasons for timetabling pre-enrolment; if it were left entirely to student choice there is no guarantee that a feasible timetable could be constructed and, secondly, more and more emphasis on opening access to universities dictates that students with busy lives need to know timetables before choosing optional parts of the course. Many universities use a phased approach which is a combination between pre- and post-enrolment. More work is required to understand the issues involved, and where, what and how search techniques and indeed what measures of optimisation can be used.

It is clear that the improvement of solutions will come about through the combination of high-level heuristics and optimisation techniques. The research challenge is therefore identified as the requirement for detailed studies of how the aims, objectives and practicalities of timetabling within institutions interlink.

### 3.2 Building the Institutional Model

As with examination timetabling, the timetable construction process can be broken down into a series of information and solution modelling steps. Even more so than with the examination problem, this process is complicated. As stated, this is related to the number of interested parties and diversity of the data requirements. Attempts have been made to provide a general framework to aid this situation. For example, work has been carried out proposing a generic architecture for the production of a timetable by examining the full range of procedures and the associated characteristics [75]. Also, in [53], a framework was presented allowing the researcher to combine many different solution methods in arbitrary ways in the solution of a single problem. Such contributions have



provided an important platform upon which we can build. A more complete description to enable understanding of the specific needs of the modelling process is required. The following impacts on a number of key issues.

In the case of course timetabling, information modelling can be broken into data, constraint and course structure modelling with solution modelling being dominated by factors related to optimisation and evaluation. Although it is an important issue, algorithmic modelling is not discussed here because the focus of this discussion is concerned with highlighting the high level challenges that need to be addressed if the gap between theory and practice is to be closed. In many respects, the key to narrowing this gap in relation to course scheduling is related to the modelling of the entire problem, thus identifying where and when in the process search techniques may be of use.

**Information Modelling.** In terms of information modelling, the main difference from examination timetabling is the much more incomplete nature of the data, with the requirements [65,75] being much more substantial. Data are required on events, course structures, the estate and the lectures/instructors availability and expertise. From the author's experience, it is evident that a combination of poorly implemented information strategies and reluctance of staff within the sector has led to a position where this information is difficult to obtain. This situation inevitably leads to significant changes in the timetable formulation at the beginning of the period in which it is required. Work has been carried out on ensuring that a changed solution is as close as possible to the initially modelled solution after changes in the original definition. For example see [65].

In many instances, expert timetablers have dealt with the initial construction by adopting a series of high-level heuristics. For example some institutions use a centralised approach initially, timetabling a percentage of the required events in a percentage of the available centrally 'owned' rooms, thus allowing individual schools/departments to 'fill in the blanks' in the remaining rooms or indeed in departmentally 'owned' rooms [56]. Many such high-level heuristics are used within institutions during the construction process, little of which (to the author's knowledge) have been reported in the literature. In general, these relate to space usage and decomposition within both the information and solution modelling process. This emphasises the fact that an important challenge for the research community is therefore to review real applications of course scheduling techniques and software with the aim of identifying the major themes which will facilitate the construction of robust initial solutions. High level heuristics need to be identified, analysed and modelled in terms of constraints and evaluation. In general these usually relate to student and staff preference and space usage.

**Course Structure Modelling.** Modelling the course structure is a difficult and important aspect of the information modelling process. This aspect is completely unnecessary in the examination counterpart. Course timetabling raises a variety of issues relating to when staff/rooms are available and what events should be timetabled with which others. The latter of these issues becomes more difficult when, as discussed earlier, it is dictated that a timetable must be ready before student enrolment. The research challenge is therefore in identifying easy

intuitive ways of representing constraints. Attempts have been made to specify a standard timetabling data format that is complete and universally applicable [28,43,73]. This work needs to be extended and made more readily available to enable users to identify and model constraints, thus allowing the interface between users and researchers to become better defined.

Another important issue is the division of students attending a lecture into sub-events such as tutorial classes. In examining this in detail a number of key issues are explored. Consider the case involving the separation of students enrolled on a particular course into tutorial classes. Consider, also, a lecture event which has  $x$  students. If the preferred size of tutorials is  $y$ , then it is trivial to calculate that  $x/y$  tutorial slots are required. The interesting research issue considered here, however, is in what way to split the  $x$  students into groups while ensuring that maximum flexibility is introduced into the timetable: i.e., what are the best combinations of students to be timetabled in which slots. In addition this must be done in a manner to allow room usage to be maximised while ensuring that students are allocated throughout the week with cognisance taken of their existing commitments on events related to other courses. This is often done manually by allowing students to self-select particular slots from a set of pre-established time slots. In the course timetabling literature, the majority of influential work on course sectioning (sometimes termed ‘splitting’) has concentrated on timetabling courses, where lectures, tutorials and laboratories, etc., are not distinguished between each other [6,27,41,42,55]. Apart from a few notable exceptions [45], courses or groups of students are subdivided into groupings for the purpose of offering student choice as opposed to reflecting the structure of events which constitute the structure of the course. The objective is normally related to balancing the size of the groups while offering students maximum choice, this enabling them to enrol on their choice of modules.

Within the UK in particular, universities subdivide students in line with course structures. The main problem with this current definition of course splitting is that sub-events do not inherit parental clashing constraints [11], apart from where a lecture event is subdivided. There is also some work dealing with students sectioning problems dated back to the 1980s [9,55]. Once again, this work is different from what we are considering here, where students are divided into sub-groups as opposite to multi-groups. More recently, fuzzy algorithms have been used [5] to cluster students in large classes into groups which may later lead to the fewest possible conflicts in timetables. Beyrouthy et al. [11] considered the problem of splitting in relation to space objectives by investigating splitting of courses of same type event into sub-events of that type for the purpose of fitting into particular room profiles. During the years little has been done on partitioning the students into actual sub-events as dictated by the course structure. In [45], metaheuristics are proposed to address the availability-based laboratory/tutorial timetabling problem (ALTP). This offers a very promising platform for further exploration into the automatic constructing of timetables while providing a solution which assigns students to the ‘best’ timeslot based on a defined week range. It should be noted that in doing so, it is important that

the needs of all parties are addressed. This raises the interesting concept of how an attained solution should be measured. When producing a course timetable within an institution, it is important that the timetable produced is seen to be fair and equitable to all interested parties. The challenge to research is investigation of these and other information modelling issues. This will be further discussed in the next section.

Another aspect of course structure modelling is related to the timetabling of associated events together. It is important to provide the ability to link particular events under the notion of course structure and schedule them as a ‘package’. This concept is similar to Kemp chains in examination timetabling [78]. This macro-event scheduling process will allow the basic building blocks of the course timetabling problem to be sustained throughout the process. This approach has the advantage of reflecting organisational and course make-up. In addition it may be possible to decide which events/courses have similarities and can be linked together when timetabling based on individual or indeed groups of characteristics. For example, pathways within a particular school could be timetabled together at the same time using the same departmental space. This mimics the construction process already in existence within an institution where the overall timetable is broken into a number of sub-units which are timetabled at a particular time by a particular person. This subdivision or decomposition of the timetabling is a challenging research aspect which needs further investigation. Macro-events may be based on a combination of course structure and clusters. Academic timetable problems tend to show signs of clustering related to the organisational structure. For instance modules from a School of Mathematics will clash with other modules from that school. Further to that, those modules will tend to clash with other science subjects such as physics and chemistry. What is required is a way of splitting such problems into smaller sub-problems in such a way that any crossover between events in different sub-problems is kept to a minimum.

### 3.3 Solution Modelling

Within the context of developing and delivering an institution-wide timetable, it must be clear what the optimisation issues are and how they are to be measured. The measurement of optimisation itself is quite different from the measure needed for the examination problem. There is sometimes a view in the research community that it is possible to define the course timetabling problem by simply altering the optimisation function used within the examination timetabling problem. However, this formulation does not define how institutions view the quality measure of a particular course timetabling solution. Institutions are interested in a combination of room usage, staff and student satisfaction. The first of these is measurable by multiplying occupancy by frequency: e.g., how many students use a room how often. The measurement of utilisation is an average of multiplication of occupancy and frequency over a set 40 hour week. Staff satisfaction is measured by the extent to which teaching duties can be ‘bunched’ together leaving time for research and other activities. In many cases, academic staff members insist on the concept of a ‘research day’. As a further advantage, it is often considered advantageous

if undesirable hours can be identified and minimised per member of staff. This is termed here as the ‘share bad hours’ heuristic and is an example of a new soft constraint to be considered when optimising the construction and improvement of an institutional course timetable. Student satisfaction can be measured by the spread of events and the availability of choice within a particular course structure. As already mentioned, ‘best spread’ has quite a different meaning in this context. A number of other issues are relevant to the overall construction problem but not the optimisation problem: e.g., staff satisfaction can further be measured by the ease at which information is gathered from them.

As previously stated, in many cases optimisation is sacrificed for the sake of getting a solution which is workable: e.g., the definition of a ‘good’ solution is driven by the need to have any solution based on a subset of the actual event types which are required [59]. This has the effect of meaning that a feasible solution is judged at an early stage in the construction process as opposed to answering the question as to whether or not the solution is actually workable: e.g., can all additional events not timetabled be accommodated after student enrolment. When students arrive and populate the skeleton structure of the timetable, solutions to individual problems of over-subscription are obtained through negotiation and compromise. The overriding factor which makes the entire process workable is the fact that currently universities utilise on average about 30 percent of their space effectively [51,52]. One explanation for this is that space utilisation is low because of the inherent flexibility within the timetable i.e. staff and students have a lot of choice. Unfortunately, this is not always the case as timetabling concerns rate highly in both student and staff surveys [57]. Further evidence of the inflexible nature of the course timetable is the fact that universities are not able to accommodate more students easily or indeed plan new or change existing course delivery. The author’s view is very much like that of Carter [41], namely that more work needs to be completed to understand the relationship between space usage, staff flexibility and student choice. It is therefore essential that metrics are produced to measure the effectiveness of timetables from all perspectives.

It is suggested that the optimisation function used to measure the quality of the problem solution must be constructed in such a manner as to take in the multi-criteria associated with each area. Whereas optimisation is relatively easily defined for examination scheduling, it is difficult to define for course scheduling. From the author’s experience, it can be defined as a balance between keeping all the stakeholders happy: e.g., student choice, staff flexibility and room usage. Therefore, to aid with the automation of the task, the construction and optimisation of the solution must take into consideration three distinct areas as an absolute minimum. In addition, in evaluating a given solution to the course timetabling problem within an institution, the users need to understand the situation in terms of the outcomes of individual constraints associated with all identified areas. The multi-objective approach has received significant recent [29,18,47] interest with respect to timetabling and, with respect to course timetabling, will be able to better express and illustrate the features of a solution to a problem.

## 4 Conclusion

This paper outlines the major challenges which face those researchers working in the area of university exam and course timetabling. While not trying to exhaustively reference the literature, detail is provided of the relevant research in both areas. The challenges are presented from the perspective of the author's experience and experience of working closely with the educational sector. The intention is to stimulate debate in the literature by providing opinion based on practical implementations. The aim is the improvement of techniques and hence software tools available to the sector to help with this most difficult and time consuming aspect of university administration.

In relation to examination scheduling the identified challenges to researchers in the area include the following:

- (i) New datasets becoming available on a regular basis encompassing more real-world requirements.
- (ii) The development of robust techniques which are able to deal with the information poor environments within which examination timetables are often developed.
- (iii) Investigation of a reformulation of the problem, including new hard and soft constraints which better reflect the real-world environment.
- (iv) Identification and comparison of key dataset characteristics and potential linkages with the likely best search approach to be taken.
- (v) The investigation of all aspects of solution quality in the provision of the 'best' solution for the institution.
- (vi) The exploration of new search technologies in establishing how developed systems can be made more general.
- (vii) Investigation of how to incorporate user interface design with the inherent complexity of the problem.
- (viii) Wide-ranging investigation of different neighbourhood structures and fitness landscape within the context of real-world problem solving environments.

In relation to course timetabling, the following research themes are highlighted:

- (i) Investigation of techniques to deal with the distributed, information poor environment in which course timetables are produced.
- (ii) Standardisation of datasets, constraints and modelling languages influenced by real-world scenarios.
- (iii) Investigation of the role in user interaction in the design of decision support system for course timetabling.
- (iv) Investigation of the need for the reformulation and modelling of the problem. It should be need that this represents a far greater challenge within the context of course timetabling than it does for examination timetabling.
- (v) Identification and adaptation of high-level policies and practices that are employed by administrators within institution to construct initial solutions.

- (vi) Experimentation related to heuristic approaches to subdivision of events.
- (vii) Investigation of the effect of pre- and post-enrolment production of the timetable on the approaches taken to optimisation: e.g., penalty used.
- (viii) Undertake an investigation into the delivery of more sophisticated models which capture the complexity and multi-objective nature of timetable evaluation in the real world.
- (ix) Investigation of the important linkage between space usage and flexibility within the academic timetable.
- (x) Investigation of approaches involving decomposition and ‘macro event’ timetabling.

In summary, this paper has outlined a number of significant research challenges which provide a rich area for research into automated search methodologies for educational timetabling. Moreover, by addressing these demanding research issues, the scientific community will be taking a step towards closing the gap between theory and practice which has existed for so long.

## References

1. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M., McCollum, B.: A tabu based large neighbourhood search methodology for the capacitated examination timetabling problem. *Journal of Operational Research Society* (accepted for publication), Advance online publication (September 13, 2006) (to appear, 2007), doi:10.1057/palgrave.jors.2602258
2. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja–Orlin’s large neighbourhood search approach for examination timetabling. *OR Spectrum* 29, 351–372 (2007)
3. Abdullah, S., Burke, E.K., McCollum, B.: An investigation of variable neighbourhood search for the course timetabling problem. In: 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA, New York, July 2005, pp. 413–427 (2005)
4. Abdullah, S., Burke, E.K., McCollum, B.: Using a randomised iterative improvement algorithm with composite neighbourhood structures for course timetabling. In: Doerner, K.F., Gendreau, M., Greistorfer, P., Gutjahr, W.J., Hartl, R.F., Reimann, M. (eds.) *Springer Operations Research. Computer Science Interfaces Book Series*, Springer, Berlin (2006)
5. Amintoosi, M., Haddadina, J.: Feature selection in a fuzzy student sectioning algorithm. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004. LNCS*, vol. 3616, pp. 147–160. Springer, Heidelberg (2005)
6. Amintoosi, M., Haddadnia, J.: Feature selection in a fuzzy student sectioning algorithm. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004. LNCS*, vol. 3616, pp. 147–160. Springer, Heidelberg (2005)
7. Asmuni, H., Burke, E.K., Garibaldi, J.M., McCollum, B.: Fuzzy multiple ordering criteria for examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004. LNCS*, vol. 3616, pp. 334–353. Springer, Heidelberg (2005)
8. Asmuni, H., Burke, E.K., Garibaldi, J.M., McCollum, B.: A novel fuzzy approach to evaluate the quality of examination timetabling. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006. LNCS*, vol. 3867, pp. 327–346. Springer, Heidelberg (2007)

9. Aubin, J., Ferland, J.A.: A large scale timetabling problem. *Computers and Operations Research* 16, 67–77 (1989)
10. Bardadym, V.A.: Computer aided school and timetabling: the new wave. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 22–45. Springer, Heidelberg (1996)
11. Beyrouthy, C., Burke, E.K., Landa-Silva, J., McCollum, B., McMullan, P., Parkes, A.J.: The teaching space allocation problem with splitting. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006*. LNCS, vol. 3867, pp. 228–247. Springer, Heidelberg (2007)
12. Beyrouthy, C., Burke, E.K., Landa-Silva, J., McCollum, B., McMullan, P., Parkes, A.J.: Understanding the role of UFOs within space allocation (Abstract). In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, August 2006, pp. 359–364 (2006)
13. Burke, E.K., Bykov, Y., Newall, J.P., Petrovic, S.: A time-predefined local search approach to exam timetabling problems. *IIE Transactions* 36, 509–528 (2004)
14. Burke, E.K., Bykov, Y., Petrovic, S.: A multi-criteria approach to examination timetabling. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 118–131. Springer, Heidelberg (2001)
15. Burke, E.K., Carter, M. (eds.): *PATAT 1997*. LNCS, vol. 1408. Springer, Heidelberg (1998)
16. Burke, E.K., De Causmaecker, P. (eds.): *PATAT 2002*. LNCS, vol. 2740. Springer, Heidelberg (2003)
17. Burke, E.K., De Causmaecker, P., Petrovic, S., Vanden Berghe, G.: Metaheuristics for handling time interval coverage constraints in nurse scheduling. *Applied Artificial Intelligence* 20, 743–766
18. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, G.: The state of the art of nurse rostering. *Journal of Scheduling* 7, 441–499 (2004)
19. Burke, E.K., Eckersley, A., McCollum, B., Petrovic, S., Qu, R.: Identifying potential similarity measures between exam timetabling problem for a case based reasoning system. In: *The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, MISTA, Nottingham, August 2003, pp. 120–136 (2003)
20. Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S., Qu, R.: Using simulated annealing to study behavior of various exam timetabling data sets. In: *MIC 2003. 5th Meta-heuristics International Conference*, Kyoto (August 2003)
21. Burke, E.K., Elliman, D.G., Ford, P.H., Weare, R.F.: Examination timetabling in British universities – a survey. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 76–90. Springer, Heidelberg (1996)
22. Burke, E., Erben, W. (eds.): *PATAT 2000*. LNCS, vol. 2079. Springer, Heidelberg (2001)
23. Burke, E.K., Kendall, G., McCollum, B., McMullan, P., Newall, J.: Optime: integrating research expertise with institutional requirements (Software demonstration). In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, August 2006, pp. 510–515 (2006)
24. Burke, E.K., Kendall, G., McCollum, B., McMullan, P., Newall, J.: A preference based measurement of optimization. *Internal eventMAP Technical Report eMAP/2006/02a*
25. Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyperheuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Meta-Heuristics*, ch. 16, pp. 457–474. Kluwer, Dordrecht (2003)

26. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9, 451–470 (2003)
27. Burke, E.K., Kingston, J.H., de Werra, D.: Applications to timetabling. In: Gross, J., Yellen, J. (eds.) *The Handbook of Graph Theory*, pp. 445–474. Chapman and Hall/CRC Press, Boca Raton, FL (2004)
28. Burke, E.K., Kingston, J., Pepper, P.: A standard data format for timetabling instances. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 213–223. Springer, Heidelberg (1998)
29. Burke, E.K., Landa Silva, J.D.: The influence of the fitness evaluation method on the performance of multiobjective optimisers. *European Journal of Operational Research* 169, 875–897 (2006)
30. Burke, E.K., McCollum, B., McMullan, P.: Examination timetabling: a new formulation (Abstract). In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 373–375 (August 2006)
31. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper heuristic for educational timetabling problems. *European Journal of Operational Research* 176, 177–192 (2007)
32. Burke, E.K., Newall, J.P.: A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation* 3.1, 63–74 (1999)
33. Burke, E.K., Newall, J.: Enhancing timetable solutions with local search methods. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 195–206. Springer, Heidelberg (2003)
34. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of Operations Research* 129, 107–134 (2004)
35. Burke, E.K., Newall, J.P., Weare, R.F.: A memetic algorithm for university exam timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996)
36. Burke, E.K., Newall, J., Weare, R.F.: A simple heuristically guided search for the timetable problem. In: Alpaydin, E., Fyfe, C. (eds.) *EIS 1998*. *Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems*, University of La Laguna, Spain, pp. 574–579. Academic, New York (1998)
37. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002)
38. Burke, E.K., Petrovic, S., Qu, R.: Case based heuristic selection for timetabling problems. *Journal of Scheduling* 9, 99–113 (2006)
39. Burke, E.K., Ross, P. (eds.): *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153. Springer, Heidelberg (1996)
40. Burke, E.K., Trick, M.A. (eds.): *PATAT 2004*. LNCS, vol. 3616. Springer, Heidelberg (2005)
41. Carter, M.W.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 64–84. Springer, Heidelberg (2001)
42. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
43. Chand, A.: A constraint based generic model for representing complete university timetabling data. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 125–150. Springer, Heidelberg (2005)
44. Coloni, A., Dorigo, M., Maniezzo, V.: Metaheuristics for high school timetabling. *Computational Optimisation and Applications* 9, 275–298 (1998)



45. Corne, D.W., Kingston, J.: Addressing the availability-based laboratory/tutorial timetabling problem with heuristics and metaheuristics. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 136–140. Springer, Heidelberg (2003)
46. Cumming, A., Paechter, B., Rankin, R.C.: Post-publication timetabling. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 107–108. Springer, Heidelberg (2001)
47. Deb, K., Pratap, A., Agarwal, S., Meyrivan, T.: A fast and elitist multi-objective genetic algorithm. *IEEE Transactions on Evolutionary Computation* 6, 182–197 (2002)
48. <http://www.diegm.uniud.it/satt/projects/EduTT>
49. Dimopoulou, M., Miliotis, P.: Implementing a university course and examination timetabling system in a distributed environment. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 148–151. Springer, Heidelberg (2001)
50. EPSRC/ESRC Document Review of Research Status of Operational Research in the UK (2004)
51. Geller, S.: Timetabling at the University of Sheffield, UK – hardening the incremental approach to timetable development. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 499–500. Springer, Heidelberg (2005)
52. HEFCE: Estates management statistics project. Technical Report. Higher Education Funding Council for England (March 1999), Report 99/18. [http://www.hefce.ac.uk/pubs/hefce/1999/99\\_18.htm](http://www.hefce.ac.uk/pubs/hefce/1999/99_18.htm)
53. Kingston, J.H., Yin-Sun Kynn, B.: A software architecture for timetable construction. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 342–350. Springer, Heidelberg (2001)
54. Konstantinow, G., Coakley, C.: Use of genetic algorithms in reactive scheduling for course timetable adjustments. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 521–522. Springer, Heidelberg (2005)
55. Laporte, G., Desroches, S.: The problem of assigning students to course section in a large engineering school. *Computers and Operations Research* 13, 387–394 (1986)
56. McCollum, B.: The implementation of a centrally computerised timetabling system in a large British civic university. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 237–254. Springer, Heidelberg (1998)
57. McCollum, B.: 2003–2004 Academic timetabling: analysis of staff and student perception. Internal eventMAP Report eMAP04/02/01
58. McCollum, B.: Bridging the gap between research and practice: university timetabling in the real world – KEYNOTE. In: Proceedings of the 47th Annual Operational Society Conference, OR47, Chester (September 2005)
59. McCollum, B., McKillop, M., McMullan, P.: Course scheduling: the division of lecture events into tutorials. Internal eventMAP Technical Report eMAP/2006/02b
60. McCollum, B., McMullan, P., Newall, J., Lane, J.P.: A workable scheduling algorithm. In: The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications, MISTA, Nottingham, pp. 570–572 (August 2003)
61. McCollum, B., Newall, J.: Introducing Optime: Examination Timetabling Software. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 485–490. Springer, Heidelberg (2001)
62. Merlot, L.T.G., Borland, N., Hughes, B.D., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 207–231. Springer, Heidelberg (2003)

63. Muller, T., Barak, R.: Interactive timetabling: concepts, techniques and practical results. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 58–72. Springer, Heidelberg (2003)
64. <http://www.metaheuristics.org>
65. Muller, T., Rudova, H., Bartak, R.: Minimal perturbation problem in course timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 126–146. Springer, Heidelberg (2005)
66. Ozan, E., Alkan, A.: Timetabling using a steady state genetic algorithm. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 104–106. Springer, Heidelberg (2003)
67. Paquete, L., Stützle, T.: Empirical analysis of tabu search for the lexicographic optimisation of the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 413–420. Springer, Heidelberg (2003)
68. Petrovic, S., Burke, E.K.: Educational timetabling. In: Leung, J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, pp. 45-1–45-23. Chapman and Hall/CRC Press, Boca Raton, FL (2004)
69. Petrovic, S., Bykov, Y.: A multiobjective optimisation technique for exam timetabling based on trajectories. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 179–192. Springer, Heidelberg (2003)
70. Petrovic, S., Patel, V., Yang, Y.: Examination timetabling with fuzzy constraints. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 313–333. Springer, Heidelberg (2005)
71. Piechowiak, A., Ma, J., Mandiau, R.: An open interactive timetabling tool. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 34–50. Springer, Heidelberg (2005)
72. Qu, R., Burke, E.K., McCollum, B., Merlot, L.G.T., Lee, S.Y.: The state of the art of examination timetabling. Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham
73. Reis, L.P., Oliveira, E.: A language for specifying complete timetable problems. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 322–341. Springer, Heidelberg (2003)
74. Rossi-Doria, O., Samples, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., Stutzle, T.: A comparison of the performance of different metaheuristics on the timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 329–351. Springer, Heidelberg (2003)
75. Rubio, R.G., Munoz, D.P.: A timetable production system architecture for course and exams. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 567–570. Springer, Heidelberg (2005)
76. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
77. Socha, K., Knowles, J., Samples, M.: A max–min ant system for the university course timetabling problem. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) *Ant Algorithms*. LNCS, vol. 2463, pp. 1–13. Springer, Heidelberg (2002)
78. Thompson, J., Dowsland, K.: A robust simulated annealing based examination timetabling system. *Computers Operations Research* 25, 637–648 (1998)
79. The Times Higher Educational Supplement, 4 (March 10, 2006)

# Very Large-Scale Neighborhood Search Techniques in Timetabling Problems

Carol Meyers<sup>1</sup> and James B. Orlin<sup>2</sup>

<sup>1</sup> Operations Research Center, Massachusetts Institute of Technology,  
77 Massachusetts Avenue, E40-130, Cambridge, MA 02139, USA

<sup>2</sup> Sloan School of Management, Massachusetts Institute of Technology,  
77 Massachusetts Avenue, E53-363, Cambridge, MA 02139, USA  
{carol,jorlin}@mit.edu

**Abstract.** We describe the use of very large-scale neighborhood search (VLSN) techniques in examination timetabling problems. We detail three applications of VLSN algorithms that illustrate the versatility and potential of such algorithms in timetabling. The first of these uses *cyclic exchange neighborhoods*, in which an ordered subset of exams in disjoint time slots are swapped cyclically such that each exam moves to the time slot of the exam following it in the order. The neighborhood of all such cyclic exchanges may be searched effectively for an improving set of moves, making this technique computationally reasonable in practice. We next describe the idea of *optimized crossover* in genetic algorithms, where the parent solutions used in the genetic algorithm perform an optimization routine to produce the ‘most fit’ of their children under the crossover operation. This technique can be viewed as a form of multivariate large-scale neighborhood search, and it has been applied successfully in several areas outside timetabling. The final topic we discuss is *functional annealing*, which gives a method of incorporating neighborhood search techniques into simulated annealing algorithms. Under this technique, the objective function is perturbed slightly to avoid stopping at local optima, while neighborhood search techniques help provide an effective search of the feasible space.

## 1 Introduction

### 1.1 Timetabling Problems

The scheduling of classes and examinations is a key practical problem that is faced by nearly all schools and universities. Substantial effort has been devoted to developing effective timetabling procedures over the last thirty to forty years. The problems tackled by such procedures include *course timetabling*, in which a set of exams is to be scheduled over a set of time periods, and *classroom timetabling*, where a set of courses must be scheduled over the length of an entire semester.

Timetabling problems are often complicated by numerous constraints; for instance, in the examination timetabling problem, students should not be scheduled to take two exams at the same time. These constraints are typically divided

into [hard constraints](#), which must not be violated (in the course timetabling problem, a hard constraint might be that no teacher is scheduled to teach two classes at once), and [soft constraints](#), which possess a penalty for being violated (in the examination timetabling problem, a soft constraint might be to minimize the number of students who take two exams back-to-back). Because of the number and variety of constraints, such timetabling problems typically constitute NP-hard problems that are quite difficult to solve manually. This in turn has led to an increased emphasis on finding effective [local search](#) timetabling algorithms.

Recent surveys on automated timetabling (see [\[22,24,25,44\]](#)) illustrate the wide array of methods that have been applied to timetabling problems. Traditional techniques tested in timetabling include [greedy algorithms](#) [\[35\]](#), which fill up the timetable one event at a time and resolve conflicts by swapping exams, and a reduction to the [graph coloring](#) problem [\[39\]](#), where events are associated with vertices of a graph and edges with potential conflicts. More modern heuristics include [tabu search](#) [\[21\]](#) and [simulated annealing](#) [\[20,28,31\]](#), which use techniques inspired by evolutionary biology; [stochastic local search](#) algorithms [\[19,48\]](#), where nonimproving solutions are permitted with progressively decreasing probability; [tabu search](#) heuristics [\[2,27,43\]](#), where a list of recently visited timetables are forbidden to be visited; and [constraint programming](#) approaches [\[26\]](#), which are based on applying declarative logic programming systems to constraint satisfaction problems.

In this paper, we address the application of very large-scale neighborhood search techniques (see Section [1.2](#)) to timetable scheduling problems, including one approach based on genetic algorithms (Section [3](#)) and one that resembles simulated annealing (Section [4](#)). Neighborhood search has long been used in timetable scheduling, from the swap (2-opt) techniques used in the direct approaches to the variety of forms of neighborhood search used in genetic algorithms. However, the area of [very large-scale neighborhood search](#) has only recently been investigated with respect to timetable scheduling [\[1,14,34\]](#) (see Section [2](#)). We believe there are many untapped possibilities for useful algorithms in this context.

## 1.2 Very Large-Scale Neighborhood Search

[Local search](#) algorithms (also known as [neighborhood search](#) algorithms) are a class of algorithms that start with a feasible solution and attempt to find an improving solution in the [neighborhood](#) of the current solution. The neighborhood structure may be defined in a variety of ways, typically so that all solutions in the neighborhood of the current solution satisfy a set of prescribed criteria. In [small neighborhoods](#), the size of the neighborhood under consideration is extremely large (typically, exponential) in the size of the problem data, making it impractical to search such neighborhoods explicitly.

A [very large-scale neighborhood search](#) (VLSN) algorithm is one that searches over a very large neighborhood, giving an improving solution in a

amount of time. Such algorithms tend to search over the neighborhood rather than explicitly, since the quantity of solutions precludes performing an exhaustive search.

There are three main categories of very large-scale neighborhood search algorithms that are outlined in [6]. The first of these is *implicit* methods, which partially search an exponentially large neighborhood by using heuristics. The second kind are *network flow* methods, which use network flow techniques to search over the neighborhood and identify improving neighbors. The third main category consists of neighborhoods based on *reductions* of NP-hard problems that are solvable in polynomial time. Ahuja et al. [6,7] provide a thorough exposition of the algorithms in these categories in their surveys on the topic.

Very large-scale neighborhood search techniques have been applied to a wide range of problems in combinatorial optimization. These include the traveling salesman problem [29,36,40], the quadratic assignment problem [8], vehicle routing problems [3,30], the capacitated minimum spanning tree problem [11], the generalized assignment problem [51,52], and parallel machine scheduling problems [4]. In several of these problems, the VLSN search algorithms give the strongest known computational results, making the development of such algorithms desirable in practice.

The design of a successful VLSN search algorithm depends on the choice of an appropriate neighborhood function and the development of an effective heuristic method to search the neighborhood for improving solutions. VLSN search techniques may also be *integrated* within the framework of other heuristic methods, such as tabu search [33,34] and scatter search [42], to provide further computational improvements. See [6,7] for a comprehensive discussion of techniques for developing strong VLSN search algorithms.

### 1.3 Contributions of This Paper

We describe three applications of very large-scale neighborhood search techniques to timetabling problems. For simplicity, we consider the timetabling problem in each of these instances, but our approaches can be modified to apply to classroom timetabling problems as well.

In Section 2, we describe the *cyclic exchange* neighborhood and how it may be applied to timetabling problems. In this neighborhood, an ordered subset of exams in disjoint time slots are swapped in a cyclic fashion such that each exam moves to the time slot of the exam following it in the order. We consider recent applications of the cyclic exchange neighborhood in the timetabling literature, and relations to other neighborhood search techniques in timetabling.

We discuss the idea of *implicit* search in genetic algorithms in Section 3. In an optimized crossover, the parent solutions used in the genetic algorithm perform an optimization routine to produce the ‘most fit’ of their children under the crossover operation. This can be viewed as a form of very large-scale neighborhood search, where the neighborhood is defined over *all* of the

parent solutions. We discuss problems for which the optimized crossover has been applied, and how a heuristic for optimized crossover could be incorporated into genetic algorithms for timetabling problems.

In Section 4, we review a new metaheuristic algorithm known as [VLSNS](#) that combines neighborhood search techniques with a type of simulated annealing algorithm. This algorithm allows the application of very large-scale neighborhood search techniques within an annealing framework, which was not previously practical due to the random selection of solutions in simulated annealing. We discuss how this algorithm has the potential to be very useful in timetable scheduling problems, on which simulated annealing algorithms have performed well in the past.

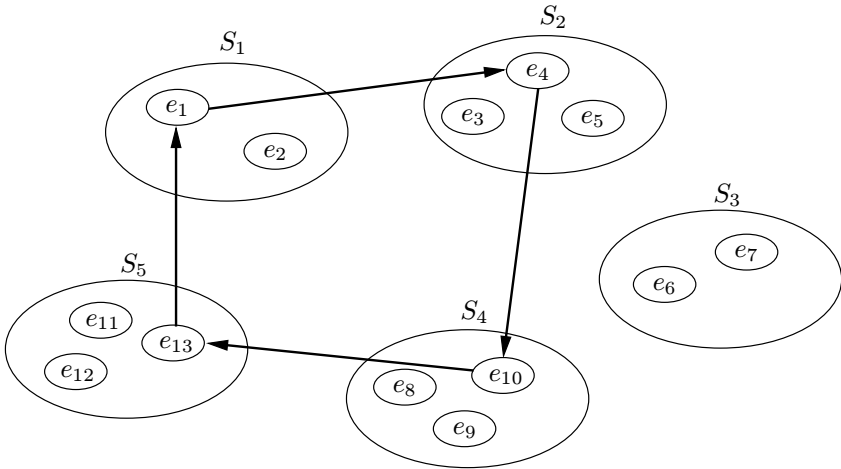
## 2 Cyclic Exchange Neighborhood

### 2.1 Definition

The cyclic exchange neighborhood is defined for partitioning problems. We present the problem here in terms of scheduling a set of exams over a collection of time periods, where potential conflicts between the exams are implicitly encoded in the objective function. However, it should be noted that this neighborhood extends to any problem that can be expressed in terms of partitioning the members of one set, so long as the cost of a partition is the sum of the cost of its parts.

Let  $E = \{e_1, e_2, \dots, e_n\}$  be a set of  $n$  exams, and let  $P = \{p_1, p_2, \dots, p_m\}$  be a set of  $m$  time periods in which we wish to schedule the exams. Suppose that  $S = \{S_1, S_2, \dots, S_m\}$  is a [partition](#) of the exams in  $E$  into  $m$  sets, such that each exam belongs to exactly one set in  $S$ , and each set  $S_i$  corresponds to the collection of exams scheduled in period  $p_i$ . Let  $c(S)$  denote the cost of solution  $S$ . We assume that any conflicts between students and exams are implicitly encoded in the objective function  $c(S)$ , so that [any](#) valid partitioning of the exams represents a feasible solution to the problem. This is similar to the approach taken by Abdullah et al. [1, 2]. Consider a sequence  $e_{i_1}, e_{i_2}, \dots, e_{i_k}$  of exams in  $E$  such that exam  $e_{i_j}$  is contained in set  $S_j$ , for each  $j$ . Suppose we switch exam  $e_{i_j}$  from set  $S_j$  to set  $S_{j+1}$ , for all  $j = 1, \dots, k-1$ , and we switch exam  $e_{i_k}$  into set  $S_1$ . We call such an operation a [cyclic exchange](#). We can also think of the exams as forming a [cycle](#)  $e_{i_1} - e_{i_2} - e_{i_3} - \dots - e_{i_k} - e_{i_1}$ , such that each exam switches to having the time slot of the exam following it in the cycle. An illustration of a cyclic exchange is given in Figure 1. In the figure, the sequence  $e_1 - e_4 - e_{10} - e_{13}$  of exams forms a cycle; exam  $e_1$  switches from  $S_1$  to  $S_2$ , exam  $e_4$  switches from  $S_2$  to  $S_4$ , exam  $e_{10}$  switches from  $S_4$  to  $S_5$ , and exam  $e_{13}$  switches from  $S_5$  to  $S_1$ . The set  $S_3$  is not included in the cyclic exchange, so its exams are not changed.

In the case where  $k = 2$ , this operation is equivalent to the [2-opt](#) operation, where a single pair of exams switch time slots. Neighborhoods defined over the 2-opt operation have been studied previously in the timetabling community by Alvarez-Valdez et al. [13], Colorni et al. [27], and Schaerf [43], among others. If instead we do not require exam  $e_{i_k}$  to move into set  $S_1$ , then we call the operation a [cyclic exchange](#), which can be described by the path of exams  $e_{i_1} - e_{i_2} - e_{i_3} - \dots - e_{i_k}$ .



**Fig. 1.** The cyclic exchange neighborhood

We can show mathematically that path exchanges may be modeled as a special case of cyclic exchanges, by adding dummy nodes as appropriate [11].

We define the *cyclic exchange neighborhood* of solution  $S$  as all partitions  $T = \{T_1, T_2, \dots, T_m\}$  that can be obtained from the sets  $\{S_1, S_2, \dots, S_m\}$  via a cyclic exchange operation. The size of this neighborhood is exponential in  $m$ , the number of periods; for a fixed value of  $m$ , the total number of cyclic neighbors of a given solution is  $O(n^m)$ . Since the size of this neighborhood is enormously large, the neighborhood structure will only be useful in practice if we have an effective search method for finding improving solutions. Fortunately, Thompson and Psaraftis [50] and Ahuja et al. [10,11] have identified several methods of finding such solutions.

## 2.2 Searching the Cyclic Exchange Neighborhood

We use the concept of an *improvement graph*, introduced in Thompson and Orlin [49] and further examined by Thompson and Psaraftis [50]. Rather than explicitly searching over each possible solution in the neighborhood, the improvement graph allows us to efficiently search the neighborhood for improving solutions. This helps reduce the amount of required computations.

For a feasible partition  $S = \{S_1, S_2, \dots, S_m\}$  of the exams, the improvement graph  $G(S)$  is a directed graph with  $n$  nodes, each corresponding to one of the exams in  $e_1, e_2, \dots, e_n$ . The arc  $(e_i, e_j)$  represents the transferring of exam  $e_i$  from the subset  $S[i] \in S$  that contains it to the subset  $S[j] \in S$  containing exam  $e_j$ , with exam  $e_j$  becoming unassigned. More formally, if we let  $S[i]$  denote the subset in  $S$  containing exam  $e_i$ , we can define the edge set as  $\{(e_i, e_j) \mid S[i] \neq S[j]\}$ , with the interpretation of each arc as previously described. The cost of arc  $(e_i, e_j)$  is set to  $c_{ij} = c(\{e_i\} \cup S[j] \setminus \{e_j\}) - c(S[j])$ . This is exactly equal to the cost of adding exam  $e_i$  to set  $S[j]$  and unassigning exam  $e_j$  from  $S[j]$ .

We say a cycle  $W$  in  $G(S)$  is *feasible* if the exams in  $E$  that correspond to the nodes in  $W$  are all scheduled in different time slots in  $S$ . (In other words, for every pair of nodes  $e_i$  and  $e_j$  in  $W$ , we have  $S[i] \neq S[j]$ .) Thompson and Orlin [49] showed that there exists a one-to-one correspondence between cyclic exchanges in  $S$  and subset-disjoint directed cycles in  $G(S)$ ; most importantly, they both have the

This result suggests that to effectively search the cyclic exchange neighborhood, we need only to identify *feasible* cycles in the improvement graph. Unfortunately, although the problem of finding a general negative cost cycle is solvable in polynomial time [9], the problem of finding a negative cost subset-disjoint cycle is NP-hard [47,49]. However, Thompson and Psaraftis [50] and Ahuja et al. [11] have identified effective heuristic algorithms that produce negative cost subject-disjoint cycles quickly in practice. Thompson and Psaraftis's heuristic begins by initially searching for only small negative cost subset-disjoint cycles (i.e., 2-cycles or 3-cycles), and uses a variable depth approach to increase cycle length and cost improvement. Although their algorithm generates and searches only a portion of the graph  $G(S)$ , it was found to be effective in practice. Ahuja et al.'s heuristic is a modification of the label-correcting algorithm for the shortest path problem, which restricts every path found by the label-correcting algorithm to being a subset-disjoint path. They found that on test instances, the time to identify a negative cost cycle was less than the time needed to construct the improvement graph.

Hence, the idea of an improvement graph can be efficiently exploited to allow searching of the cyclic exchange neighborhood. Using the algorithms of Thompson and Psaraftis and Ahuja et al., improving solutions in the neighborhood can be found successfully for many problem types. This suggests that the cyclic exchange neighborhood may be a valuable network structure to consider in solving timetabling problems.

### 2.3 Cyclic Exchange in the Timetabling Literature

Cyclic exchange neighborhoods have been investigated only recently in the timetabling literature. For this reason, we believe this is a potentially fruitful area for research in timetabling. We now outline a couple of the studies in which the cyclic exchange neighborhood has been incorporated.

Abdullah et al. [1] initiated the first study of the cyclic exchange neighborhood in examination timetabling problems. To identify negative cost subset-disjoint cycles, they used the heuristic of Ahuja et al. [11]. They additionally introduced an exponential Monte Carlo acceptance criterion (see [15]) for accepting nonimproving moves. In this way, their algorithm is less likely to become stuck at a local optimum. Tests of the algorithm against other timetabling algorithms on common benchmarks showed that the performance of their algorithm is comparable to that of the best currently known timetabling algorithms. Most recently, the authors have extended the algorithm by incorporating a tabu search heuristic with the selection of improving moves [2], which has produced even more promising results.



Jha [34] has also recently studied the usefulness of cyclic exchange neighborhoods in timetabling problems. His algorithm uses a dynamic programming approach to identify negative cost subset-disjoint cycles. He also combines the cyclic exchange heuristics with a tabu search framework, to avoid the problem of halting at local optima. In terms of implementation, he found that the VLSN–tabu search combination produced robust solutions in a reasonable amount of time. Compared to approaches using integer programming or neighborhood search alone, he found that the VLSN–tabu search algorithm performed better on larger test instances.

Together, these two studies suggest that the combination of cyclic exchange techniques with other suitable timetabling heuristics can make for especially strong algorithms. Whether the methods used are Monte Carlo acceptance techniques or tabu search, the combination of the VLSN methodology with the existing algorithms can be used to produce a more effective algorithm overall.

## 2.4 Relation to Other Techniques in the Literature

As mentioned in Section 2.1, the  $2\text{-opt}$  operation is a special case of the cyclic exchange operation, where each cycle has length equal to 2. This is occasionally referred to as the  $2\text{-opt}$  operation, since it consists of swapping the time slots of a pair of exams. The  $2\text{-opt}$  neighborhood is defined as the set of all possible solutions that can be reached from a given solution by performing a single  $2\text{-opt}$  move.

Many papers in the timetabling literature have used neighborhood search over the  $2\text{-opt}$  neighborhood to refine timetabling solutions, though not necessarily using that name and most often in conjunction with other techniques. Alvarez-Valdes et al. [13] used  $2\text{-opt}$  moves combined with tabu search in finding solutions for timetabling problems in the Spanish school system. Schaefer [43] combined tabu search and the randomized nonascendent method with  $2\text{-opt}$  neighborhood search techniques in solving high school timetabling problems. Colorni et al. [27] used  $2\text{-opt}$  techniques along with simulated annealing, tabu search, and genetic algorithms for problems from Italian high schools; they found the combination of genetic algorithms with tabu search to be especially powerful. Carter [23] addresses the scheduling of classes at the University of Waterloo by decomposing the problem into several subproblems, which are then solved using a greedy procedure including  $2\text{-opt}$  moves.

It should be noted that while  $2\text{-opt}$  moves can be done efficiently in the improvement graph (since there are only  $O(n^2)$  possible such moves), they are inherently a lot weaker than cyclic exchange moves. For this reason, it would be interesting to apply the cyclic exchange neighborhood to the same classes of problems. This presents a fruitful, and largely unexamined, avenue for new research.

## 3 Optimized Crossover in Genetic Algorithms

### 3.1 Overview of Genetic Algorithms

Genetic algorithms are an optimization technique based on the mechanisms of evolution and natural selection [38]. In applying genetic algorithms to

time-tabling problems, we assume (as in Section 2) that any valid partitioning of exams into a timetable  $T$  represents a feasible solution, and that potential conflicts between the exams are implicitly encoded in the objective function. (It should be noted that it is also possible to extend the following definitions to the constrained version of the problem.) There are a wide range of ways to implement genetic algorithms. We describe a classic approach.

In each iteration of a genetic algorithm, a population of solutions is maintained, which represent the current set of candidate solutions. At time  $t = 0$ , a population of timetables  $\{T_1^0, T_2^0, \dots, T_K^0\}$  is generated randomly from the set of all possible solutions. In further iterations, the population  $\{T_1^{t+1}, T_2^{t+1}, \dots, T_K^{t+1}\}$  at time  $t + 1$  is generated from the population at time  $t$  according to the  $fi$  of each of the candidate solutions  $T_i^t$  ( $i = 1, \dots, K$ ), along with crossover and mutation operations.

The  $fi$  function is a problem-specific measure of how good a timetable is. One obvious candidate for the fitness of a solution is its objective function value. (However, in problems for which calculating the objective is time-consuming, alternative methods of fitness can be formulated.) In selecting a set of candidate solutions at time  $t$  to produce the next generation at time  $t + 1$ , the algorithm begins by assessing the fitness of all timetables at time  $t$ . Next, a number of individuals from the population are randomly selected, based on a weighted randomization scheme; the ‘fitter’ a solution is, the more likely it is to be selected.

The crossover operation functions by taking two of the selected timetables  $T_i$  and  $T_j$  and combining them to form a new timetable. The selected timetables are referred to as the parent timetables, and the new timetable is called the child timetable. In what follows, we assume that the parent timetables are represented in the form  $(p_1^k, p_2^k, \dots, p_n^k)$ , where  $p_\ell^k$  represents the time period in which exam  $e_\ell$  is scheduled in timetable  $T_k$ .

The crossover operation can take several forms, of which the  $fi$  crossover is very common. In this situation, a given position  $\ell \in \{1, \dots, n - 1\}$  is selected; the child solution is created by concatenating the first  $\ell$  periods in the timetable of the first parent with the last  $n - \ell$  periods in the timetable of the second parent. Hence, if  $T_i$  and  $T_j$  are the first and second parents, their child solution will have the form  $(p_1^i, \dots, p_\ell^i, p_{\ell+1}^j, \dots, p_n^j)$ .

Another frequently used crossover scheme is the  $fi$ , where two random positions  $\ell_1$  and  $\ell_2$  ( $\ell_1 < \ell_2$ ) are selected; in this case, the child is formed by taking the periods of the first parent in the intervals  $(1, \ell_1)$  and  $(\ell_2 + 1, n)$  and the periods of the second parent in the interval  $(\ell_1 + 1, \ell_2)$ , giving a solution of the form  $(p_1^i, \dots, p_{\ell_1}^i, p_{\ell_1+1}^j, \dots, p_{\ell_2}^j, p_{\ell_2+1}^i, \dots, p_n^i)$ . Similarly, we can define  $fi$  by first generating a random number  $N$ , arbitrarily determining  $N$  crossover positions, and then creating the child by taking each odd interval from the first parent and each even interval from the second parent.

The  $fi$  operation is used to ensure diversity of the timetables generated. In this operation, a given position  $\ell$  in timetable  $T_k$  is selected with some (small) probability  $P_m$ , and exam  $e_\ell$  is reassigned from period  $p_\ell^k$  in which it is currently scheduled to another randomly selected time period. This has the

effect of ‘mutating’ the  $\ell$ th exam period from its original value. In this way, time periods that are not a part of the set of parent timetables can be present in the successive generation, which occasionally leads to better solutions.

### 3.2 Optimized Crossover

In the previous section we discussed the crossover operation in genetic algorithms. One striking feature of this method is that the crossover points are determined [\[5\]](#), and the resulting child is created [\[5\]](#). Hence occasionally the fitness of a child can deviate quite widely from the fitness of its parents. Aggarwal et al. [\[5\]](#) suggested instead choosing the [\[5\]](#) child from all possible children, building on an idea of Balas and Niehaus [\[16\]](#) in the area of graph theory.

The set of all possible children  $T_{ij}$  from two timetables  $T_i$  and  $T_j$  can be written as  $\{T_{ij} \mid p_{ij}^\ell = p_i^\ell \text{ or } p_{ij}^\ell = p_j^\ell, \text{ for all } \ell = 1, \dots, n\}$ . Thus, the period in which any exam is scheduled in  $T_{ij}$  will either be the same as the period in which it is scheduled in  $T_i$ , or else the same as the period in which it is scheduled in  $T_j$ . The problem of finding the [\[5\]](#) child is then the problem of choosing from among the  $O(2^n)$  possible children the one with the best objective function.

We can think of solving the optimized crossover problem as a type of very large-scale neighborhood search. In this case, the neighborhood is defined over a [\[5\]](#) of parent solutions, instead of a single solution. This is a somewhat unusual use of the term ‘neighborhood’, but we claim the concept is plausible since the neighborhood is well-defined. For each pair of solutions  $T_i$  and  $T_j$ , the crossover neighborhood is defined as the set of all possible children  $T_{ij}$  that can be produced from  $T_i$  and  $T_j$ . The problem of finding the [\[5\]](#) child can be viewed as that of finding the child with the best objective value in the crossover neighborhood.

The idea of optimized crossover has not been previously used in genetic algorithms for timetabling problems, and we believe it is an excellent candidate for study. In the next two sections, we detail a few of the areas in which optimized crossover has proven to be useful, followed by comments on the feasibility of the method on timetabling problems in particular.

### 3.3 Previous Applications of Optimized Crossover

Aggarwal et al. [\[5\]](#) were the first to apply the concept of optimized crossover to genetic algorithms. They studied the independent set problem, for which they gave an effective method of combining two independent sets to obtain the largest independent set in their union. This was based on a related technique of Balas and Niehaus [\[16\]](#). Their resulting genetic algorithm incorporated this optimized crossover scheme, and was shown to be superior to other genetic algorithms for the independent set problem. This approach was further verified by Balas and Niehaus [\[17\]](#).

Ahuja et al. [\[12\]](#) later extended the idea of optimized crossover to genetic algorithms for the quadratic assignment problem. They presented a matching-based optimized crossover heuristic that finds an optimized child quickly in practice.

This technique can also be applied to other assignment-type problems, as it relies on the structure of the problem rather than the objective function.

Most recently, Ribeiro and Vianna [41] have applied the idea of optimized crossover to genetic algorithms for building phylogenetic trees, which are trees showing evolutionary relationships among species with a common ancestor. Their algorithm outperforms the best algorithms currently available. Lourenço et al. [37] have also used a type of optimized crossover heuristic in their study of bus driver scheduling. They solve a set-covering subproblem to determine the best child solution; their algorithm outperforms other algorithms tested, albeit at a higher computational cost.

### 3.4 Optimized Crossover in Timetabling Problems

As mentioned in Section 3.2, for an optimized crossover to be effective in practice, it requires a method of quickly obtaining a best (or very good) child solution from two parents. The problem of finding the optimized crossover in timetable scheduling problems is unfortunately NP-hard, via a transformation from the Minimum Set Cover problem (see [32]). Hence, the best we can hope for is to find a strong heuristic for obtaining a good crossover. We now describe how this can be accomplished in timetabling problems.

The algorithm we consider here is a greedy algorithm, which starts with the two parent solutions  $T_i$  and  $T_j$ . First it randomly selects an order to consider the exams in. The algorithm proceeds through the exams in order, where for each exam  $e_k$  it places the exam in either slot  $T_i^k$  or  $T_j^k$  according to which one gives the smallest increase in the objective function. The result will be a scheduling of exams that (hopefully) gives a low objective value. (Many other variations in the greedy algorithm are possible.)

This algorithm will perform quickly in practice, as once the ordering is decided upon there are only two choices for each of the exams. The quality of the solutions produced by the algorithm may vary depending on the quality of the ordering.

Thus we have given a heuristic for solving the optimized crossover problem in genetic algorithms for timetabling problems. Though this method has not been tested in a timetabling context, the strong results obtained for the crossover method in other problems (see [5,12]) make it an attractive avenue to pursue in the area of timetabling.

## 4 Functional Annealing

### 4.1 The Functional Annealing Algorithm

The functional annealing method is a relatively new metaheuristic for combinatorial optimization problems. Proposed by Sharma and Sukhapesna [45,46], it combines the attractive components of both a neighborhood search method and a simulated annealing algorithm. As simulated annealing algorithms have been extensively examined in the timetabling literature (see, for instance, [19] and [48]), we believe this method should be greatly appealing to the timetabling

community. In this subsection and the next two, we outline the functional annealing algorithm and its properties, followed by a discussion of applying functional annealing techniques to timetabling problems in particular.

The main idea of the functional annealing method is to introduce a stochastic element into the objective function, while employing an efficient neighborhood search strategy. The stochastic element is given in terms of an  $\epsilon$ -perturbation, which tends to the original objective as the number of iterations increases. The perturbed objective allows the algorithm to escape efficiently from local optima, while the neighborhood search heuristic provides for a more effective search of the feasible space.

We now describe the algorithm more formally, following the structure of Sukhapesna [46]. Suppose we are given a 0–1 discrete optimization problem (such as a timetabling problem), with a cost function  $c(x)$  and a neighborhood  $N(x)$  for each element  $x$  in the set  $F \subseteq \{0,1\}^n$  of feasible solutions. We let  $c(x, w) = c(x) + w'x$  be our annealing function, where  $w$  is a random vector in  $\mathbb{R}^n$  with independent and identically distributed elements. The distribution of  $w$  is determined by a control parameter  $U$ , such that  $w$  approaches zero as  $U$  approaches zero. We assume we are given a sequence  $\{U_k\}$  of such control parameters, such that  $U_k > 0$  for all  $k \geq 0$  and  $\lim_{k \rightarrow \infty} U_k = 0$ . Thus, the longer the algorithm runs, the less stochasticity there is in the objective function. The functional annealing algorithm is described in Figure 2.

**algorithm** *functional annealing*

**begin**

    choose an initial solution  $x_0$  in  $F$ ;

    set  $k = 0$ ;

**while** stopping criteria are not met, **do**

        generate a vector  $w_k$  such that  $w_k(i) = e_k(i)$  if  $x_k(i) = 1$  and  $w_k(i) = -e_k(i)$  if  $x_k(i) = 0$ , where  $e_k$  is distributed exponentially with mean  $U_k$ ;

        using a neighborhood search algorithm, find a neighboring solution  $y \in N(x) \cup \{x\}$ , such that  $c(y, w_k) \leq c(x, w_k)$ ;

        set  $x_{k+1} = y$ ;

        set  $k = k + 1$ ;

**end;**

**end;**

**Fig. 2.** The functional annealing algorithm

As can be seen from the algorithm, the random vector  $w_k$  is always chosen so that the perturbation attempts to make the current solution worse than its neighbors, which has the effect of forcing the algorithm to move away from its current solution. (Note that the algorithm is allowed to stay at the same solution between iterations, but if there is a better solution, it can be shown that this solution will be found with probability 1.) Moreover, the magnitude

of the perturbation vector  $w_k$  is such that the greater the number of iterations, the smaller the influence of the perturbation. Hence for small values of  $k$ , the algorithm behaves similarly to a search for a random neighbor, and for large enough values of  $k$ , the algorithm behaves more like a deterministic neighborhood search algorithm.

One of the appealing features of using a neighborhood search strategy in tandem with the functional annealing approach is that the algorithm will not spend multiple iterations at a solution that is not a local optimum, in contrast to the standard simulated annealing algorithm. Another item of note is that in the case of a linear objective, the algorithm is equivalent to a problem where the data is perturbed to avoid lingering at local optimal solutions (see [18]).

## 4.2 Properties of the Algorithm

A natural question one might have about the functional annealing algorithm is whether it is guaranteed to reach the set of optimal solutions. Indeed, Sharma and Sukhapesna [45,46] have shown that the algorithm is guaranteed to attain the set of optimal solutions with probability 1, provided that the neighborhood search algorithm is such that at any given step each improving solution is chosen with positive probability. Moreover, the expected number of iterations needed to reach an optimal solution is finite.

With respect to the choice of improving neighbors, the authors consider a  $fi$  strategy, in which improving solutions in the neighborhood are selected with equal probability. If no improving neighbor is found, then the current solution is kept for the next iteration. They show that the chance of exiting from the current solution under such a strategy is less than that of simulated annealing, and for large numbers of iterations the exiting probability is about  $|N(x)|$  times less than that of simulated annealing. Thus the functional annealing algorithm is better in theory than simulated annealing in terms of becoming stuck at local optima. They also show that a  $fi$  strategy is also guaranteed to reach the set of optimal solutions with probability one, though the time to find a solution takes longer than with the first improvement strategy.

Sharma and Sukhapesna [45,46] give a thorough computational study of functional annealing algorithms applied to the quadratic assignment problem. They show that the functional annealing algorithm performs better than both simulated annealing and neighborhood search algorithms on instances of the problem, confirming the earlier theoretical results. This improvement holds regardless of the size of the instance being considered. They also show that the best improvement strategy tends to outperform the randomized first improvement strategy on small instances, while on larger instances the difference is less pronounced. They conclude by showing that incorporating a  $fi$  strategy along with the functional annealing algorithm gives the strongest computational results overall.

### 4.3 Functional Annealing and VLSN Search

Functional annealing provides a way to integrate neighborhood search techniques within the framework of annealing methods. Since the only condition on the neighborhood search algorithm is that it should be able to produce an improving solution in the neighborhood in a reasonable amount of time, we can easily apply existing VLSN techniques to the functional annealing algorithm.

For instance, the cyclic exchange neighborhood (see Section 2) can be incorporated into the functional annealing algorithm. This neighborhood is too large to be of practical interest with the standard simulated annealing algorithm, since the simulated annealing algorithm functions by comparing the performance of random solutions in the neighborhood. The cyclic exchange neighborhood is so large that there is no reason to believe that a random solution will perform well. This problem is alleviated in the functional annealing approach, because it does not rely on the generation of purely random solutions in the neighborhood.

Sharma and Sukhapesna [45,46] incorporated the cyclic exchange neighborhood in their analysis of functional annealing algorithms for the quadratic assignment problem. They found that in small problem instances, algorithms using the cyclic exchange neighborhood consistently outperformed algorithms based on a 2-opt structure (see Section 2.4). The results for large problem instances were less dramatic.

### 4.4 Functional Annealing and Timetabling Problems

Functional annealing techniques can be applied to timetabling problems in the same way that simulated annealing algorithms are currently used. (See [19] and [48] for details on the implementation of simulated annealing algorithms in timetabling problems.) Typically, the only restriction on the format of the solutions is that they are represented in such a way that the neighborhood search subroutine can be performed adequately. In the case of the cyclic exchange neighborhood, for instance, we could use the problem structure previously outlined in Section 2.

A main advantage of the functional annealing algorithm is that it allows us to use very large-scale neighborhood search techniques along with annealing algorithms, which have already been used successfully in timetabling problems (see [44] for a survey). For this reason, we believe that this algorithm has a potential to be very valuable to the timetabling community.

## 5 Concluding Remarks

In this paper, we have discussed one application and two potential applications of very large-scale neighborhood search techniques in examination timetabling problems. The applications range from one that has been used before in timetabling problems (the *2-opt* neighborhood search algorithm), to one that has been widely used in contexts other than timetabling (the *2-opt* neighborhood search algorithm in genetic algorithms), to a relatively new concept that we believe has a great potential for timetabling problems (the *cyclic exchange* neighborhood search algorithm).

Although these applications are presented in the context of examination timetabling, the techniques are general enough to apply to a wide range of timetabling problems. It is our hope that the timetabling community will make use of these techniques and incorporate them into further studies in the timetabling literature. Based on the existing work, we believe that very large-scale neighborhood search techniques may be very useful in the design of new timetabling algorithms.

This work was supported in part through NSF Grant DMI-0217123.

## References

1. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja–Orlin’s large neighbourhood search approach for examination timetabling. *OR Spectrum* 29, 351–372 (2007)
2. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M., McCollum, B.: A tabu-based large neighbourhood search methodology for the capacitated examination timetabling problem. *Journal of the Operational Research Society* (to appear, 2007)
3. Agarwal, R., Ahuja, R., Laporte, G., Shen, Z.: A composite very large-scale neighborhood search algorithm for the vehicle routing problem. In: *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, ch. 49, Chapman and Hall/CRC, Boca Raton, FL (2003)
4. Agarwal, R., Ergun, Ö., Orlin, J., Potts, C.: Solving parallel machine scheduling problems with variable depth local search. Working Paper, Operations Research Center, MIT, Cambridge, MA (2004)
5. Aggarwal, C., Orlin, J., Tai, R.: Optimized crossover for the independent set problem. *Operations Research* 45, 226–234 (1997)
6. Ahuja, R., Ergun, Ö., Orlin, J., Punnen, A.: A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, 75–102 (2002)
7. Ahuja, R., Ergun, Ö., Orlin, J., Punnen, A.: Very large-scale neighborhood search: theory, algorithms, and applications. Working Paper, Operations Research Center, MIT, Cambridge, MA (2006)
8. Ahuja, R., Jha, K., Orlin, J., Sharma, D.: Very large-scale neighborhood search for the quadratic assignment problem. Working Paper, Operations Research Center, MIT, Cambridge, MA (2002)
9. Ahuja, R., Orlin, J., Magnanti, T.: *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Upper Saddle River, NJ (1993)
10. Ahuja, R., Orlin, J., Sharma, D.: Very large-scale neighborhood search. *International Transactions in Operational Research* 7, 301–317 (2000)
11. Ahuja, R., Orlin, J., Sharma, D.: Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming* 91, 71–97 (2001)
12. Ahuja, R., Orlin, J., Tiwari, A.: A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research* 27, 917–934 (2000)
13. Alvarez-Valdes, R., Martin, G., Tamarit, J.: Constructing good solutions for the spanish school timetabling problem. *Journal of the Operational Research Society* 47, 1203–1215 (1996)



14. Avella, P., D'Auria, B., Salerno, S., Vasil'ev, I.: Computational experience with very large-scale neighborhood search for high-school timetabling. Working Paper, Research Center on Software Technology, Università del Sannio, Italy (2006)
15. Ayob, M., Kendall, G.: A Monte Carlo hyper heuristic to optimise component placement sequencing for multi-head placement machine. In: Proceedings of the 4th International Conference on Intelligent Technologies, Chiang Mai, Thailand, pp. 132–141. Institute for Science and Technology Research and Development, Chiang Mai University (2003)
16. Balas, E., Niehaus, W.: Finding large cliques in arbitrary graphs by bipartite matching. In: Clique, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, pp. 29–53. AMS (1996)
17. Balas, E., Niehaus, W.: Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics* 4, 107–122 (1998)
18. Bertsimas, D., Tsitsiklis, J.: *Introduction to Linear Optimization*, Athena Scientific, Belmont, MA (1997)
19. Bullnheimer, B.: An examination scheduling model to maximize students' study time. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 78–91. Springer, Heidelberg (1998)
20. Burke, E.K., Elliman, D., Weare, R.: A hybrid genetic algorithm for highly constrained timetabling problems. In: Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, pp. 605–610. Morgan Kaufmann, San Mateo, CA (1995)
21. Burke, E.K., Newall, J., Weare, R.: A memetic algorithm for university exam timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996)
22. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002)
23. Carter, M.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 64–82. Springer, Heidelberg (2001)
24. Carter, M., Laporte, G.: Recent developments in practical examination timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996)
25. Carter, M., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
26. Cheng, C., Kang, L., Leung, N., White, G.: Investigations of a constraint logic programming approach to university timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 112–129. Springer, Heidelberg (1996)
27. Coloni, A., Dorigo, M., Maniezzo, V.: Metaheuristics for high-school timetabling. *Computational Optimization and Applications* 9, 275–298 (1998)
28. Corne, D., Ross, P., Fang, H.: Fast practical evolutionary timetabling. In: Fogarty, T.C. (ed.) *Evolutionary Computing*. LNCS, vol. 865, pp. 251–263. Springer, Heidelberg (1994)
29. Deineko, V., Woeginger, G.: A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical Programming* 87, 255–279 (2000)
30. Ergun, Ö.: New neighborhood search algorithms based on exponentially large neighborhoods. Ph.D. Dissertation, Massachusetts Institute of Technology (June 2001)

31. Fang, H.: Genetic algorithms in timetabling and scheduling. Ph.D. Dissertation, University of Edinburgh (September 1994)
32. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York (1979)
33. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics* 65, 223–253 (1996)
34. Jha, K.: Very large-scale neighborhood search heuristics for combinatorial optimization problems. Ph.D. Dissertation, University of Florida (June 2004)
35. Junginger, W.: Timetabling in Germany – a survey. *Interfaces* 16, 66–74 (1986)
36. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. *Operations Research* 21, 498–516 (1973)
37. Lourenço, H., Paixão, J., Portugal, R.: Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science* 35, 331–343 (2001)
38. Mitchell, M.: *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA (1996)
39. Neufeld, G., Tartar, J.: Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM* 17, 450–453 (1974)
40. Punnen, A., Kabadı, S.: Domination analysis of some heuristics for the traveling salesman problem. *Discrete Applied Mathematics* 119, 117–128 (2002)
41. Ribeiro, C., Vianna, D.: A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path relinking. In: *Proceedings of the 2nd Brazilian Workshop on Bioinformatics, Macaé, Brazil*, pp. 97–102. SBC (2003)
42. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Working Paper, Department of Computer Science, University of Copenhagen, Denmark (2005)
43. Schaerf, A.: Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans* 29, 368–377 (1999)
44. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
45. Sharma, D., Sukhapesna, S.: Functional annealing technique for very large-scale neighborhood search. Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI (in preparation)
46. Sukhapesna, S.: Generalized annealing algorithms for discrete optimization problems. Ph.D. Dissertation, University of Michigan (September 2005)
47. Thompson, P.: Local search algorithms for vehicle routing and other combinatorial problems. Ph.D. Dissertation, Massachusetts Institute of Technology (May 1988)
48. Thompson, J., Dowland, K.: A robust simulated annealing based examination timetabling system. *Computers and Operations Research* 25, 637–648 (1998)
49. Thompson, P., Orlin, J.: The theory of cyclic transfers. Working Paper OR 200-89, Operations Research Center, MIT, Cambridge, MA (1989)
50. Thompson, P., Psaraftis, H.: Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research* 41, 935–946 (1993)
51. Yagiura, M., Ibaraki, T.: Recent metaheuristic algorithms for the generalized assignment problem. In: *Proceedings of the 12th International Conference on Informatics Research for Development of Knowledge Society Infrastructure, Kyoto, Japan*, pp. 229–237. IEEE Computer Society Press, Los Alamitos, CA (2004)
52. Yagiura, M., Iwasaki, S., Ibaraki, T., Glover, F.: A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discrete Optimization* 1, 87–98 (2004)

# Measurability and Reproducibility in University Timetabling Research: Discussion and Proposals

Andrea Schaerf and Luca Di Gaspero

Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica  
Università degli Studi di Udine  
via delle Scienze 208, I-33100, Udine, Italy  
{schaerf,l.digaspero}@uniud.it

**Abstract.** In this paper, we first discuss the level of compliance for timetabling research to two important research qualities, namely measurability and reproducibility, analyzing what we believe are the most important contributions in the literature. Secondly, we discuss some practices that, in our opinion, could contribute to the improvement on the two aforementioned qualities for future papers in timetabling research.

For the sake of brevity, we restrict our scope to university timetabling problems (exams, courses, or events), and thus we leave out other equally important timetabling problems, such as high-school, employee, and transportation timetabling.

## 1 Introduction

Thanks mainly to the PATAT conference series, researchers on timetabling problems have recently started to meet regularly to share experiences and results. This situation has the positive effect of generating both a common language and a common spirit that is the base ground for cross-fertilization of research groups in the timetabling community.

However, according to what we have seen at the recent PATAT conferences, the road for timetabling to become a well-established research community is still long. The main issue, in our opinion, is that most timetabling papers tend to describe the authors' specific problem and *ad hoc* solution algorithm without taking enough care of either the *measurability* or the *reproducibility* of the results. The reader is thus 'left alone' to judge the quality of the paper, and to understand what can be learned from it.

This issue is, to some extent, common to all the experimental areas of computer science and operations research, as clearly explained by Johnson in his seminal paper [17]. Nevertheless, we believe that this is particularly true in timetabling research, probably because of its shorter standing as a scientific community.

Regarding measurability (or comparability), we believe that several 'research infrastructures' are necessary in order to create the ground for truly measurable results. Specifically, they range from common formulations, to benchmark instances, to instance generators, to solution validators, and others. Related to it,

but somewhat complementary, is the issue of reproducibility. To this aim, beside the features just mentioned, it would be also necessary to create the conditions for sharing code and/or executables among researchers.

In this paper, we try to describe the main contributions with respect to these crucial qualities of experimental research in timetabling, and we also present some personal opinions on how to proceed to improve on them. For the sake of brevity, we restrict our scope to university timetabling problems (exams, courses, or events), and we leave out other equally important timetabling problems, such as high-school, employee and transportation timetabling. Nevertheless, to some extent, the proposed guidelines can have a broader application to all timetabling domains.

In detail, we first survey what, in our opinion, are the most important steps that have been pursued so far in timetabling research in terms of either measurability or reproducibility of results (Section 2). Secondly, we propose our personal ‘best practices’ for improving these two qualities in the timetabling research (Section 3). Our aim is to encourage both the authors to write research papers of high level in these important aspects and the reviewers to demand it when judging a paper.

## 2 Significant Contributions

In this section, we review the most significant contributions to the aim of creating the ground for the development of high quality measurable and reproducible research in timetabling. We first discuss the ‘standard’ problem formulations, the benchmark instances (datasets), and the related file formats adopted. Next, we move to the comparison methods proposed, such as competitions between algorithms and statistical tools. Finally, we discuss the issue of the objective validation of the proposed results.

### 2.1 Problem Formulations and Benchmark Instances

It is well known that timetabling problems vary not only from country to country, but also from university to university, and even in different departments of the same university the problem is not quite the same (see, e.g., [27]).

Nevertheless, throughout the years it has been possible to define common underlying formulations that could be used for the comparison of algorithms. In fact, a few basic formulations have become standards *de facto*, as they have been used by many researchers. Needless to say, standard formulations allow the researchers to compare their results and to co-operate for the solution. Furthermore, in some cases algorithms developed for more complex *ad hoc* formulations can be adapted to the basic standard ones so as to assess their objective quality.

For the Examination Timetabling problem (ETTP), Carter et al. [7] propose a set of formulations which differ from each other based on some components of the objective function. Carter also makes available a set of benchmark instances [6] extracted from real data, which represent a large variety of different situations. Formulations and benchmarks by Carter have stimulated a large body

of research, so that many researches (see, e.g., [4,8,15]) have adopted one of the formulations of Carter (or a variant of them, creating a new standard as well), tested on the benchmarks, and also added new instances. For more complex formulations, additional data have been added by other researchers, in an arbitrary but realistic way. At present, all available instances and the corresponding best results (only up to 2003, unfortunately) are published on the Web [20].

We call as Lecture Timetabling problem (LTTP), the problem of weekly scheduling a set of single lectures (or events). This problem differs from course timetabling (discussed below) because the latter is based on courses composed by multiple lectures, whereas lectures are independent. In fact, when a course is given in multiple lectures per week, some cost components are related to the way the lectures are placed in the week. In contrast, this concept is totally absent in LTTP. The LTTP differs also from ETTP because it has completely different objectives (e.g., no isolated event vs. spreading exams).

The LTTP has been discussed in [26] and it was the subject of the International Timetabling Competition ITC-2002 [23]<sup>1</sup>. The formulation proposed for ITC2002 has also become quite standard, and many researchers have used it for their work (see, e.g., [9,19]). Twenty artificial instances were generated for the competition, and they are available from the ITC-2002 web page. In addition, a few others have been proposed (and made available via web) in [28].

Regarding the Course Timetabling problem (CTTP), which, as mentioned above, consists of the weekly scheduling of the lectures of a set of university courses, unfortunately no standard formulation has emerged from the community so far. To our knowledge, the only formulation available on the Web [14] together with a set of instances is the one proposed by ourselves in [16], along with four instances coming from the real cases (suitably simplified and made anonymous) in our university.

## 2.2 Data Format

For all the problems mentioned above, an important issue for the spreading of a formulation in the community is the data format. For all the formulations discussed above, the data format used is an ad hoc fixed-structure text-only one. For example, for ITC-2002 the input data comes in a single file containing the scalar values (number of events, rooms, room features, and students), followed by the elements of the input arrays, one per line. The output format follows the same idea. For the ETTP the input format is also rather ‘primitive’, with a fixed grammar and no formatting tags. Unfortunately, for this problem no output format has been specified in the original paper and the associated web site.

The use of fixed-structure formats makes it easier to parse the input from any computer language, and for any (naive) programmer, but may be more difficult

---

<sup>1</sup> Such a problem has often been referred to as CTTP (or UCTP), where C stands for course (and U for university); but we believe this is quite misleading, because it deals with isolated lectures/events, rather than courses composed by many lectures. Therefore we prefer for this problem the name LTTP.

to be maintained and checked. For example, it happened that Carter’s ETPP instances were replicated incorrectly on other web sites. This was due to the presence of a few ‘newline’ characters added in the files, that led to different (less constrained) instances. As documented in detail in [25], this unfortunate episode has caused the publication of confusing results in some papers, and would have been avoided if a structured format had been used.

On the other hand, a structured format, such as XML, would be more suitable in terms of flexibility, extensibility, and maintenance, but it might hinder the work of researchers who cannot use it because of limited programming capabilities.

A few structured formats have been proposed in the literature, such as STTL [5,18] and TTML [22]. In [12], the authors go even beyond the language, proposing a multi-layer architecture for the specification and the management of timetabling problems. To our knowledge, however, these proposals have received limited attention so far in the academic community (although they are used in practical applications). This is probably due to the fact that researchers have normally little interest in the advantages of a structured language, and they prefer the quick-and-simple text-only version.

### 2.3 Comparison Methods and Competitions

The fair comparison of different algorithms and heuristics is well known to be a complex problem, and it has no simple and straightforward solution. In fact, in order to assess that an algorithm is ‘better’ than another it is necessary to specify not only the instances used, but also on which features they are compared under (e.g., quality of the objective function, success rate, speed, ...). The question gets even more complicated in presence of randomized/stochastic algorithms, which add a degree of non-determinism in the solution process.

For ITC-2002, the solution algorithms (provided as executables) were granted a maximum CPU time for their execution (based on a CPU benchmark, about 500 seconds on a recent computer) and they were evaluated only on the value of the objective function upon the 20 proposed instances. Unfeasible solutions were not considered, so that, in order to be admitted to the evaluation, participants had to find a feasible solution for all instances.

For stochastic algorithms, the participant had to ensure that their solver could produce the same solution when checked by the organization (by providing the seed of the random generator). In this situation, it is not clear how to apply the CPU time restriction and the choice of the organization was to grant the maximum time *for each single trial*. This was done to ensure reproducibility, although it had a drawback. The participants could take advantage of what we call the *Mongolian horde* approach: ‘Run as many trials as you can and report only the best of all of them’. It is worth mentioning that in order to provide against the excessive use of the Mongolian horde approach, the competition organizers tested the best few algorithms also on unseen instances, and indeed the results were found to be broadly in line with the known instances.

To our knowledge, the ITC-2002 has been the sole attempt in this respect, and a new timetabling competition is scheduled for the second half of the 2007. All other comparisons are based on results published in the literature, which however often report only part of the necessary information (running times, number of trials, ...).

## 2.4 Result Validation

When some results are claimed in a research paper, the reader (or, more importantly, the reviewer) generally has to trust the author without any actual proof on the results. Although the possibility that the author is deliberately claiming a fake result is rare, cases in which the claimed results turned out to be wrong are relatively frequent. They are normally due to bugs in the code or misunderstandings in the formulation of the problem, typically the objective function.

For example, for the Graph Coloring problem, for the famous benchmark instance DSJC125.5 a 12-coloring solution has been claimed in 2002 (see [10]), whereas it has been successively proved that the minimum number of colors is 17.

Therefore the validation of the results claimed is clearly an important step toward the full reproducibility of the results. For the LTTP, in the ITC-2002, the validation of the results was done directly by the organizers, who asked all the participants to supply an executable that accepts a set of fixed command-line arguments.

For ETTP no validation tool has been available until very recently, and therefore validation has been based only on voluntary peer-to-peer interaction based on exchanges of solutions and values. Just before PATAT-2006, Rong Qu created a new web site [24] that allows the visitors to download an executable that validates ETTP solutions (using a raw fixed-structure output format). Up to now, the executable validates only solutions for the basic version of ETTP.

For our formulation of the CTTP, we have developed a web page [14] that allows other researchers to download the problem formulation, the data format, and the benchmark instances. More importantly, everybody is allowed also to upload and validate her/his own solutions, and to insert them among the results obtained for the specific instance. All results are automatically published on the web site along with the date and other information.

## 3 Proposals

In this section, we highlight some practices that, in our opinion, could contribute to the improvement on measurability and reproducibility for future papers in timetabling research. Part of what we propose here can be found also in [17], although we try to extract the advice of Johnson that we believe is best suited to the current state of timetabling research.

### 3.1 Statistically Principled Comparison

One of the key issues of performance measurement (often underestimated) concerns the methods to deal with the random nature of many techniques for obtaining a sound comparison of the different ones. In the practice, this issue is often neglected and just some tendency indicators of the stochastic variables, like mean values (and, more seldom, also standard deviations) in  $n$  runs (with  $n \approx 10$ ), are provided. Furthermore, in a rather myopic view, these summary values are often advocated as the final word on the clear superiority of a technique over its competitors.

However, as is common knowledge in other research areas, when dealing with stochastic variables it is not correct to draw any conclusion only on the basis of single estimates, but a principled statistical analysis on the behavior of the algorithm is needed (see, e.g., [131]). Even in the simplest cases of comparison of two means, the analysis should include some kind of hypothesis testing (e.g., the  $t$ -test or the Mann–Whitney test for the parametric and the non-parametric case, respectively), that at least provides the reader with a probability measure of ‘confidence’ in the result. For more complex settings further analyses could be carried on and the statistical tool-case is plentiful of methods for correctly coping with several situations that arise in practice (see, e.g., [21]).

As an example, Birattari [2] has proposed a principled methodology for the comparison of stochastic optimization algorithms, called RACE, which comes out also as a software package for the R statistical software [3]. The RACE procedure, originally developed for the purpose of selecting the parameters of a single meta-heuristic, could be employed also in the case of the comparison of multiple algorithms by testing each of them on a set of trials. The algorithms that perform poorly are discarded and not tested anymore as soon as sufficient statistical evidence against them is collected. This way, only the statistically proven ‘good’ algorithms continue the race, and the overall number of tests needed to find the best one(s) is limited. Each trial is performed on the same randomly chosen problem instance for all the remaining configurations and a statistical test is used to assess which of them are discarded. The RACE procedure has been applied in the context of timetabling in [13].

It is worth noting that the statistical comparison of algorithms outlined in this section is based on the assumption of having full access to previous results (or, better, to the code) of the different techniques involved in the comparison. This is clearly related to the issue of reproducibility of results that, in our opinion, can be achieved by observing the guidelines described in the following.

### 3.2 Formulation, Data Format, Instances, and Results on the Web

As already mentioned, many papers in timetabling describe the modeling and the ad hoc solution of a new timetabling problem. For this kind of papers, in general we cannot expect that the authors make all the steps for obtaining full measurability and reproducibility such as, for example, publishing all the code. In fact, this would be quite a big job that would probably be too time-consuming for a researcher, beside possible employer’s concerns. Nevertheless, we believe



that there are a few actions that could contribute in these respects, which are not too expensive in terms of human work.

First, the authors must state the problem clearly and exhaustively. If this is not possible in the paper for space reasons, the full formulation should be posted in an accompanying web site. Secondly, the authors should also post in the web site all the instances considered in the study (hiding identities for privacy reasons, if necessary), along with all the necessary information accompanying them: data format, algorithms, results, and running times. Finally, the authors should post also the files containing their best solutions, so that other researcher can verify the actual results, and possibly use that solutions for further studies and improvements.

These actions would ensure comparability with the results on future research by other researchers or also by the same authors.<sup>2</sup>

### 3.3 Web-Based Problem Management System

Nowadays it is very common to see web sites that describe all aspects of either a specific problem, see e.g. [11,29], or a research area [30]. These web sites normally exhibit references to papers, people, problem formulations, and benchmark instances, and supply other information.

Web sites are surely very useful for the community, and their presence is crucial for the quality of the research. Nevertheless, we believe that there is a further step to be made to this regard. Inspired by the well-known concept of CMS (*content management system*), we envision the idea of developing what we would call PMS (*problem management system*). A PMS is a web application (rather than a web site) that should allow the users to interact with the application performing all the following tasks:

**Add results:** New results are first validated, and then possibly inserted in the database along with the time-stamp and other user-supplied information.

**Add instances:** Instances can be inserted at any moment. Researchers that are interested in the problem can be automatically informed by email of this kind of event.

**Manage instance generation:** Newly generated instances can be created automatically by users through interaction with an instance generator.

**Analyze instances and results:** Instances and results can be analyzed automatically so as to produce important indicators: constrainedness, similarity to other instances or other results, etc.

**Add general information:** People, references, links, code, and other information can be added. Links would be validated periodically in an automatic way, and broken ones can be removed. References can also be imported from other sites.

**Translate data:** Input and output data can be translated so that coherent data can be proposed in different formats to the community (including both fixed-structure and XML-based ones).

---

<sup>2</sup> Many researchers – including ourselves! – have experienced the frustration of losing their solutions (or other data) for some of the problems they have worked on.

**Organize on-line competitions:** Competitions on specific instances and with registered participants and fixed deadlines can be organized semi-automatically. Results can be reported immediately.

**Visualize:** Solutions can be visualized in graphical form to give an immediate picture of the features and the violations.

**Maintain a discussion forum:** A simple discussion forum about the problem can be maintained along with the site. Messages would be organized and displayed as in usual on-line forums (threads, date, ...).

The interesting point is that information posted through the PMS would get on-line immediately in an automatic way. Obviously, a PMS needs to provide against possible abuses and malicious behavior, and therefore some of the actions mentioned above would need the approval of the administrator before becoming effective. For most operations, this however would be just a Yes/No button, so that the administrator will hopefully operate in short time.

The PMS would also maintain historical data (through versioning systems), in such a way to be able to retrieve information eliminated by updates and deletions.

*Acknowledgements.* We wish to thank Marco Chiarandini for fruitful discussions about measurability and reproducibility of research results. We also thank the anonymous referees for their comments that helped us to improve the paper.

## References

1. Barr, R., Golden, B., Kelly, J., Resende, M., Stewart, W.: Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics* 1, 9–32 (1995)
2. Birattari, M.: The problem of tuning metaheuristics as seen from a machine learning perspective. Ph.D. Thesis, Université Libre de Bruxelles, Belgium (2004)
3. Birattari, M.: The RACE Package (April 2005)
4. Burke, E.K., Newall, J.: A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation* 3, 63–74 (1999)
5. Burke, E.K., Pepper, P., Kingston, J.: A standard data format for timetabling instances. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 213–222. Springer, Heidelberg (1998)
6. Carter, M.W.: Carter's test data (2005) (viewed March 13, 2007) (updated June 7, 2005), <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/>
7. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society* 74, 373–383 (1996)
8. Casey, S., Thompson, J.: Grasping the examination scheduling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 232–244. Springer, Heidelberg (2003)
9. Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O.: An effective hybrid approach for the university course timetabling problem. *Journal of Scheduling* 9, 403–432 (2006)

10. The website of graph coloring and its generalization (2004) (viewed March 13, 2007), <http://mat.gsia.cmu.edu/COLOR04>
11. Culberson, J.: Graph coloring page (2006) (viewed March 13, 2007) (updated March 31, 2004), <http://www.cs.ualberta.ca/~joe/Coloring/>
12. De Causmaecker, P., Demeester, P., Lu, Y., Vanden Berghe, G.: Using Web standards for timetabling. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 238–257. Springer, Heidelberg (2003)
13. Di Gaspero, L., Chiarandini, M., Schaerf, A.: A study on the short-term prohibition mechanisms in tabu search. In: ECAI-2006. Proceedings of the 17th European Conference on Artificial Intelligence, Riva del Garda, Italy, pp. 83–87.
14. Di Gaspero, L., Fontanel, A., Schaerf, A.: Educational Timetabling @UniUd (2006) (viewed March 13, 2007) (updated May 30, 2006), <http://www.diegm.uniud.it/satt/projects/EduTT/>
15. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001)
16. Di Gaspero, L., Schaerf, A.: Multi-neighbourhood local search with application to course timetabling. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 262–275. Springer, Heidelberg (2003)
17. Johnson, D.S.: A theoretician’s guide to the experimental analysis of algorithms. In: Goldwasser, M.H., Johnson, D.S., McGeoch, C.C. (eds.) Data Structures, Near Neighbor Searches, and Methodology: 5th and 6th DIMACS Implementation Challenges, pp. 215–250. American Mathematical Society, Providence, RI (2002), available from <http://www.research.att.com/~dsj/papers.html>
18. Kingston, J.H.: Modelling timetabling problems with STTL. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 309–321. Springer, Heidelberg (2001)
19. Kostuch, P.: The university course timetabling problem with a three-phase approach. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 109–125. Springer, Heidelberg (2005)
20. Merlot, L.: Public Exam Timetabling Data Sets (2005) (viewed March 13, 2007) (updated October 13, 2003), <http://www.or.ms.unimelb.edu.au/timetabling>
21. Montgomery, D.C.: Design and Analysis of Experiments, 6th edn. Wiley, New York (2005)
22. Özcan, E.: Towards an XML-based standard for timetabling problems: TTML. In: Kendall, G., Burke, E.K., Petrovic, S., Gendreau, M. (eds.) MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications, pp. 163–185. Springer, Berlin (2005)
23. Paechter, B., Gambardella, L.M., Rossi-Doria, O.: International Timetabling Competition Webpage (2003) (viewed March 13, 2007) (updated July 10, 2003), <http://www.idsia.ch/Files/ttcomp2002/>
24. Qu, R.: The exam timetabling site (2006) (viewed March 13, 2007) (updated July 8, 2006), <http://www.cs.nott.ac.uk/~rxq/ETTP.htm>
25. Qu, R., Burke, E.K., McCollum, B., Merlot, L., Lee, S.Y.: The state of the art of examination timetabling. Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham, UK (2006)
26. Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardello, L.M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquette, L., Stützle, T.: A comparison of the performance of different metaheuristics on the timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 329–351. Springer, Heidelberg (2003)

27. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
28. Socha, K., Knowles, J., Sampels, M.: A MAX-MIN Ant System for the University Timetabling Problem. In: Fieker, C., Kohel, D.R. (eds.) *Algorithmic Number Theory*. LNCS, vol. 2369, Springer, Heidelberg (2002), Data available from: <http://iridia.ulb.ac.be/~msampels/tt.data/>
29. Trick, M.: Challenge Traveling Tournament Instances, web page (2005) (viewed March 13, 2007) (updated October 22, 2006), <http://mat.gsia.cmu.edu/TOURN/>
30. The web site of the EURO working group on automated timetabling (WATT) (2002) (viewed March 13, 2007) (updated February 21, 2007), <http://www.asap.cs.nott.ac.uk/watt/>
31. Xu, J., Chiu, S., Glover, F.: Fine-tuning a tabu search algorithm with statistical tests. *International Transactions on Operational Research* 5, 233–244 (1998)

# **Employee Timetabling**

# Physician Scheduling in Emergency Rooms

Michel Gendreau<sup>1,2</sup>, Jacques Ferland<sup>1,2</sup>, Bernard Gendron<sup>1,2</sup>,  
Noureddine Hail<sup>1</sup>, Brigitte Jaumard<sup>1,3</sup>, Sophie Lapierre<sup>1,4</sup>, Gilles Pesant<sup>1,5</sup>,  
and Patrick Soriano<sup>1,6</sup>

<sup>1</sup> Interuniversity Centre for Research on Enterprise Networks,  
Logistics and Transportation (CIRRELT), Université de Montréal,  
C.P. 6128, succ. Centre-ville, Montréal, Canada, H3C 3J7

`michel.gendreau@cirrelt.ca`

<sup>2</sup> Département d'informatique et de recherche opérationnelle, Université de Montréal,  
C.P. 6128, succ. Centre-ville, Montréal, Canada, H3C 3J7

<sup>3</sup> Concordia Institute for Information Systems Engineering, Concordia University,  
1455 De Maisonneuve Blvd. W., Montréal, QC, Canada H3G 1M8

<sup>4</sup> Département de mathématiques et génie industriel, École Polytechnique de  
Montréal, C.P. 6079, succ. Centre-ville, Montréal, QC, Canada H3C 3A7

<sup>5</sup> Département de génie informatique, École Polytechnique de Montréal,  
C.P. 6079, succ. Centre-ville, Montréal, QC, Canada H3C 3A7

<sup>6</sup> Service d'enseignement des méthodes quantitatives de gestion, HEC Montréal,  
3000, chemin de la Côte-Sainte-Catherine, Montréal, QC, Canada H3T 2A7

**Abstract.** We discuss the problem of constructing physician schedules in emergency rooms. Starting from practical instances encountered in five different hospitals of the Montreal (Canada) area, we first propose generic forms for the constraints encountered in this context. We then review several possible solution techniques that can be applied to physician scheduling problems, namely tabu search, column generation, mathematical programming and constraint programming, and examine their suitability for application depending on the specifics of the situation at hand. We conclude by discussing the problems encountered when trying to perform computational comparisons of solution techniques on the basis of implementations in different practical settings.

## 1 Introduction

Constructing schedules (rosters) is not an easy task to accomplish in settings where work must be performed 24 hours per day and 7 days a week, such as in police and fire departments, or in emergency rooms of hospitals. The problem that one is faced with is to generate ‘good schedules’ that satisfy many complicated rules, including ergonomic rules as defined by Knaunth [20,21]. As mentioned by Carter and Lapierre [12], ergonomic constraints are very important in order to manage the circadian rhythm of the staff and it is critical to take them into account when building schedules.

In this paper, we focus on the problem of the scheduling of physicians in emergency rooms (ER) in health care institutions where work is continuous. It

is known that ER are a very stressful place for physicians, but it is also a great challenge for them to work in such a place. According to Lloyd et al. [24], 24.5% of physicians in Canadian ER are not satisfied with their jobs. Consequently, making a ‘good’ schedule for physicians in ER is very important. A good schedule for a physician is a schedule that satisfies a large number of the requests he or she may have regarding different issues: total amount of work to be performed, specific timing of shifts, sequencing of shifts, etc.

As already mentioned, building such schedules is quite difficult and it may take up to several weeks for a human expert to generate an acceptable solution [3]. In order to reduce time and effort, an automated approach is therefore imperative.

Besides the biological and psychological effects involved in the scheduling of physicians, one must also pay careful attention to the fairness of the schedules among physicians. This important aspect is unfortunately very difficult to address because it involves balancing the distribution of different types of shifts among physicians with respect to several criteria that often conflict.

In this paper, we give an overview of the typical constraints that may be encountered in physician scheduling by building on the lessons learned from five practical cases encountered in hospitals of the Montreal (Canada) area: Jewish General Hospital (JGH), Charles-Lemoyne Hospital (CLH), Santa-Cabrini Hospital (SCH), Sacré-Coeur Hospital (SaCH), and Côte-Des-Neiges Hospital (CNH). An important purpose of the paper is to formalize the specific constraints of these five settings into ‘generic constraints’ that could be used to describe problems in other practical contexts. We also review major approaches for solving the problem: mathematical programming, tabu search, constraint programming and column generation.

The remainder of this paper is organized as follows. In Section 2, we define more precisely the problem of scheduling physicians in ER and review the relevant literature. In Section 3, we propose the generic constraints that capture the essence of the various constraints encountered in the five physician scheduling case studies. Section 4 is devoted to solution approaches. We briefly discuss the similarities and differences between physician scheduling and nurse rostering problems in Section 5. Finally, we conclude in Section 6.

## 2 Problem Definition and Literature Review

In the health care area, there are two important types of scheduling problems that involve medical staff: nurse scheduling problems and physician scheduling problems. In the first category of problems, nurses work under collective agreements, while in the second category, there are no such rules for physicians. Moreover, in the nurse staff problem, one has to maximize their individual satisfaction and minimize the cost of salaries, whereas in the physician staff problem, one only cares about the maximization of their individual satisfaction. Despite these differences between nurse and physician problems, their formulations are not very different. Indeed, according to Rousseau et al. [27], a pure mathematical

programming approach proposed by Berrada et al. [4,5,35] for the nurse scheduling problem can successfully be applied to the physician scheduling problem.

The physician scheduling problem can be described as the preparation of a rostering for physicians for a given planning period, such that every shift of every day must be assigned to exactly one physician. To achieve this goal, we have to deal with some rules that are divided into two categories : compulsory (or hard) rules and flexible (or soft) ones. These rules are often in conflict with one another, therefore some of them have to be violated in order to have a complete schedule for all physicians. Carter and Lapierre [12] note in their investigation that some flexible rules in some hospitals might be compulsory in others and vice versa. This classification depends in general on the preferences of the hospital and on the physicians' flexibility.

The set of shifts that must be covered is specified for each day of the week. In many situations, the weekend shifts are quite different from week days shifts. In general, we have three kinds of shifts: days, evenings, and nights. A week usually begins on Monday, by the first day shift and ends Sunday with the last night shift. The planning period can be quite long (up to 6 months) or fairly short (between 2 and 4 weeks). The physicians who work in emergency rooms are divided into two categories: full-time doctors and part-time doctors. A full-time doctor works an average of 28 hours per week, part-time physician works on average between 8 and 16 hours.

The physician scheduling problem can be summarized as follows: given a set of doctors, a set of shifts and a planning period, one seeks to find fair schedules for all physicians in order to maximize their individual satisfaction.

As we have mentioned above, this problem has not received very much attention. There are, however, some software packages that have been used successfully in this context [12]:

- *Tangier Emergency Physician Scheduling Software*, by Peake Software laboratories [31];
- *Epsked*, by ByteBloc Medical Software [10];
- *Docs for Windows*, by Acme Express [1];
- *Physician Scheduler 4.0*, by Sana-Med.

These software packages have been sold to emergency departments in thousands of copies, but the research community did not benefit from the fundamental work that led to these products. The only academic works that we are aware of are some works on cyclic rostering [9,22] and some on acyclic rostering [2,3,9,11,12,15,27,32]. The solution methods developed in these references will be examined more closely in Section 4.

### 3 Physician Scheduling Problem Constraints

In this section, we propose generic forms for the constraints encountered in the five case studies mentioned in the Introduction. As we have already mentioned,



in the physician scheduling problem, we have to find a roster for every physician such that a large number of constraints are satisfied. Some constraints are applied for every physician and others only for some physicians. There are two types of constraints: hard and soft. A constraint is called *hard* if it must be satisfied; it is called *soft* if it can be violated. In this study, we have classified the constraints of the physician scheduling problem into four categories:

1. Supply and demand constraints
2. Workload constraints
3. Fairness constraints
4. Ergonomic constraints.

The first category of constraints deals with the availabilities of the physicians and the requirements of the emergency rooms that must be opened every day and 24 hours a day. The second category deals with the workload (number of hours or number of shifts) that is assigned to physicians during a week, a given period or the whole planning period. The third category controls the distribution of different kinds of shifts during the whole planning period. The fourth category of constraints covers various rules ensuring a certain level of quality for the schedules produced.

### 3.1 Supply and Demand Constraints

Two kinds of constraints are encountered in all physician scheduling problems. First, a sufficient number and variety of shifts must be staffed throughout the scheduling horizon in order to guarantee minimum coverage. Second, a given physician, according to his seniority, full/part time status, outside responsibilities, and planned vacations, is not available at all times.

**Constraint 1 (Demand).** *During the overall planning period, every shift must be performed by exactly one physician.*

Whereas in other contexts such as nurse scheduling, the number of staff members covering a shift must lie in a certain interval, for physician scheduling this number is almost always exactly one. This constraint is considered a hard constraint and it is encountered in all the hospitals listed in the Introduction. Carter and Lapierre [12] identify three variants of this situation, but we restrict our attention here to the two main ones.

1. *Uniform case:* the required number of physicians is the same for every day in a week, i.e., we have the same number of shifts for every weekday, even for Saturday and Sunday.
2. *Non-uniform case:* the required number of physicians is the same for every weekday except for Saturday and Sunday. In this case, the number of physicians required on Saturday is the same as on Sunday.

**Constraint 2 (Availability).** *During the planning period, all the requests of every physician should be satisfied. There are four types of requests:*

1. *Preassignments,*
2. *Forbidden assignments,*
3. *Vacations,*
4. *Preferences or aversions.*

Each one of these types of requests is considered a hard constraint except for the last one, which is a soft version of the first two. That last type occurs for example in the context of religious practices at JGH: some physicians want to be off for the evening and the night shifts on Friday [9].

### 3.2 Workload Constraints

This category of constraints deals with the workload (number of hours or number of shifts) that is assigned to physicians during a week, a month or the whole planning period.

**Constraint 3 (Limits on workload).** *During a given period, a physician should be assigned an amount of work that lies within a specified interval.*

*Example 1.* In the SaCH case study, a physician who is supposed to work 28 hours a week could accept to work up to 32 hours.

*Example 2.* At JGH, at most four shifts are assigned to a physician on any given week.

This constraint is common to all the hospitals we considered. It is often specified over disjoint subsets of the planning period, either because of the terms of a contract or to encourage a uniform workload. Sometimes a target workload with the interval may be given: it can be viewed as a soft constraint. Another constraint encouraging uniform workloads is the following.

**Constraint 4 (Limits on the number of shifts of the same type).** *During a given period (e.g., a month), the number of shifts of the same type that are assigned to a physician cannot exceed a certain value.*

*Example 3.* At SaCH, no physician should work more than three night shifts in a four-week period.

### 3.3 Fairness Constraints

This category of constraints ensures the fair distribution of different types of shifts among physicians with the same experience.

**Constraint 5 (Distribution of types of shifts).** *During the planning period, shifts of the same type (e.g., evening, night, weekend) should be distributed fairly among physicians with the same level of experience.*

*Example 4.* At SaCH, all physicians with more than four years of experience have to work the same number of night shifts during the planning period of six months.

*Example 5.* Again at SaCH, physicians should not work more than five weekend shifts in a four-week period. In this hospital, a working weekend can include up to three shifts.

### 3.4 Ergonomic Constraints

This is the largest and the most heterogeneous category of constraints. Various rules ensure a certain level of quality for the schedules produced and may be specified either globally for the staff or only for certain individuals. In his work on ergonomics, Knauth [20,21] has shown the impact of work schedules on the circadian rhythm of workers. He proposed several rules, which we summarize below:

- minimizing permanent night shifts;
- reducing the number of successive night shifts to a maximum of two or three;
- avoiding short intervals of time off (less than 11 hours) between two consecutive shifts;
- shift systems including work on weekends should provide some free weekends with at least two consecutive days off;
- long work sequences followed by four to seven days of mini-vacations should be avoided;
- forward rotations (day shifts followed by evening shifts followed by night shifts) are preferred;
- individual schedules with few changes over time are preferred;
- shift lengths should be adjusted according to task intensity;
- shorter night shifts should be considered;
- a very early start time for the morning shift should be avoided;
- preference should be given to flexible working time arrangements among workers.

The constraints below address some of these ergonomic concerns.

**Constraint 6 (Length of work sequences).** *The number of identical shifts (or of shifts of the same type) in a sequence of consecutive days must lie within a given interval.*

*Example 6.* In the work of Carter and Lapierre [12], there must be at least two and at most four consecutive identical shifts.

*Example 7.* At SaCH, the interval is  $[1, 4]$  for shifts in general.

*Example 8.* In each of the hospitals studied, the number of consecutive night shifts lies between one and three.

*Example 9.* At SaCH, a physician requires at least 14 days between two night shifts belonging to different work sequences. This can be recast as a constraint on the length of sequences of non-night shifts.

**Constraint 7 (Patterns of shifts).** *Over a given number of consecutive days, a set of patterns of shifts describes what a physician is allowed to do or not to do.*

*Example 10.* There must be a minimum number of hours of rest between two consecutive shifts. Consequently, certain patterns of shifts over two consecutive days are forbidden.

*Example 11.* At SaCH, a set of restrictive patterns govern weekend work. For instance, a physician working the 8AM regular shift on Saturday must also cover the 10AM trauma shift on Sunday; working the 4PM regular shift on Friday requires working the 4PM trauma shift on Saturday and the 4PM regular shift on Sunday as well.

*Example 12.* A physician should work at most one night shift in every sequence of three consecutive work shifts.

*Example 13.* A physician should not work a non-homogeneous sequence of four consecutive work shifts.

**Constraint 8 (Patterns of sequences of shifts).** *This is similar to the previous constraint, except that patterns are expressed not over a fixed number of consecutive days, but rather over a fixed number of sequences of consecutive work shifts.*

*Example 14.* At JGH, every two consecutive sequences of work shifts should satisfy the forward rotation principle.

**Constraint 9 (Patterns of sequences of a given length).** *Patterns are expressed over both the type and the length of sequences.*

This has the flavour of the previous constraint and of the first ergonomic constraint.

*Example 15.* After coming back from a vacation, no physician should work a night shift for the first two days.

*Example 16.* At SaCH, there must at least three days off after a sequence of three night shifts.

Table 1 presents a summary of these generic constraints.

**Table 1.** Generic constraints in the five hospitals studied

Constraints	CNH	CLH	JGH	SaCH	SCH
Demand	X	X	X	X	X
Availability	X	X	X	X	X
Limits on workload	X	X	X	X	X
Limits on shifts of the same type		X	X	X	X
Distribution of types of shifts	X	X	X	X	X
Length of work sequences	X	X	X	X	X
Pattern of shifts	X	X	X	X	X
Pattern of sequences of shifts			X	X	
Pattern of sequences of given length				X	

## 4 Four Optimization Techniques for the Physician Scheduling Problem

In this section, we present general descriptions of four solution techniques for the physician scheduling problem. These methods are completely different from one another, as we shall see later:

1. Mathematical programming
2. Column generation
3. Tabu search
4. Constraint programming.

### 4.1 Mathematical Programming

Beaulieu et al. [3] have proposed a mixed 0–1 programming formulation of the physician scheduling problem where the objective function is the sum of penalties associated to some constraints, called deviation constraints. This formulation was also used by Forget [15] in the context of Santa-Cabrini Hospital (SCH). In these case studies, constraints are classified in three categories: ergonomic constraints, distribution constraints and deviation constraints. After obtaining the mathematical formulation of problem under study, Beaulieu et al. [3] first considered using branch-and-bound on this formulation to find a solution, but this approach had to be dropped, unfortunately, due to the huge dimension (large number of variables and constraints) of some instances. The solution technique that was applied is a heuristic approach based on a partial branch-and-bound, instead of a complete branch-and-bound, which requires more computational time. Moreover, branch-and-bound was not applied to the original formulation, but to a modified one. Indeed, as mentioned by Beaulieu et al., it was quickly

realized that there was no feasible solution to the original formulation. This was due to the presence of some ergonomic constraints that were conflicting and led to an infeasible problem. The solution technique proposed by the authors is to solve the model with a subset of constraints which contains all hard constraints and some soft constraints that are not in conflict with each other. Afterwards, they modified some of the soft constraints and introduced them one by one in an iterative process, which can be summarized as follows [3]:

- Identify the rules that are violated in the current schedule.
- Add the corresponding constraints to the model.
- Use the branch-and-bound method to identify a new schedule, which hopefully improves over the previous one (e.g., satisfies more rules).

This process is repeated until the branch-and-bound cannot find any feasible schedule.

## 4.2 Column Generation

The column generation technique [13,26] is an exact method that relies on the decomposition principles of mathematical programming; it is usually used to solve large and complex problems, such as the cutting stock problem. This method was successfully applied to solve the nurse scheduling problem and a software called IRIS was produced [23]. In the column generation method, each new column is generated by solving an auxiliary problem (or subproblem). For instance, in the cutting stock problem, a knapsack problem is solved to find a new cutting pattern for rolls. In the nurse scheduling problem, a new column is obtained by solving a shortest path problem with resource constraints on a directed graph [33]. The resources correspond to the following constraints:

- The constraint dealing with the workload of every nurse for a given period (e.g., 2 weeks);
- The constraint that controls the vacation periods of every nurse;
- The constraint that deals with the succession of shifts of the same type;
- The constraint that is associated with the distribution of weekends.

The formulation of the master problem for the nurse scheduling problem includes the hard constraint that gives the required number of nurses for every shift of every day. Moreover, the objective function is given by the sum of penalty costs associated with the constraints not explicitly taken into account in either the auxiliary problem or the master problem.

This solution technique can be applied to the physician scheduling problem after some minor modifications. First, one can use the same auxiliary problem as for the nurse scheduling problem. Indeed, the constraints that define the resources are also present in the physician scheduling problem. Second, the constraint dealing with the requirements (number of nurses per shift), which is used in the master problem for the nurse scheduling problem, is also present in the physician scheduling problem (one physician for every shift). One then simply

has to modify the formulation of the objective function and define in it penalty costs for the remainder of the constraints that one wishes to consider.

### 4.3 Tabu Search

Tabu search is one of the most effective solution techniques for solving hard combinatorial problems. Originally proposed by Glover [19], it has been successfully applied to a wide variety of application contexts, such as vehicle routing [17], machine scheduling [29], the maximum clique problem [18], and the quadratic assignment problem [28,30]. This method has also been applied to the nurse scheduling problem [7,14], as well as the physician scheduling problem. In the case of physician staff, the solution technique was used to generate two kinds of schedules: cyclic schedules [22] and acyclic schedules [9].

Generally speaking, tabu search is a local search (LS) technique: i.e., an iterative search procedure that, starting from an initial feasible solution, progressively improves it by applying a series of local modifications. The key ingredient of any LS technique is the set of modifications (or *moves*) that it considers: the richer this set, the better the solutions that one can expect to obtain, but also the slower the method. While classical LS methods stop when they encounter a local optimum w.r.t. the modifications they allow, tabu search continues moving to the best non-improving solution it can find. Cycling is prevented through the use of short-term memory structures called *tabu lists* (see [16] for a comprehensive introduction to the topic).

Buzon's tabu search method for acyclic schedules [9] is in fact an extension and a generalization of previous work by Labbé [22]. In this approach, a solution  $S$  corresponds to a set of schedules: one for each physician. The solutions examined by the search have the property that they satisfy the demand constraints, i.e., all shifts are covered, but other constraints may be violated. The cost  $c(S)$  of solution  $S$  is the sum of the costs of all schedules in  $S$ . If there are  $n$  physicians, then the cost of a solution  $S$  is  $\sum_{p=1}^n \text{cost}(\text{Schedule}_p)$ , where  $\text{cost}(\text{Schedule}_p)$  is the cost of the schedule for physician  $p$ . The cost of a physician schedule is also the sum of all penalties that are associated with the unsatisfied constraints. There is exactly one penalty for each constraint. For example, suppose that physician  $p$  wants to work only two unbroken weekends. If the schedule associated with this physician in the current solution contains three unbroken weekends and one broken weekend, then the penalty associated with the weekend constraint would be  $(3 - 2) \cdot P_{NBW} + 1 \cdot P_{BW}$ , where  $P_{NBW}$  (respectively  $P_{BW}$ ) is a certain value associated with one extra unbroken (respectively broken) weekend. Proper values for these penalty weights are not easy to determine; unfortunately, the quality of the solution that one can expect to find is quite sensitive to them [9].

Buzon's method considers several different types of modifications to solutions (neighborhoods) of increasing complexity. The simplest one involves simply re-assigning a shift on one day to a physician currently off on that day. More complex neighborhoods involve swapping portions of schedules between two physicians. See [9] for further details.

## 4.4 Constraint Programming

Constraint programming is a solution technique that is more and more applied to various optimization and combinatorial problems. Its application to complex problems like work schedules [25] is possible for each problem in which the set of values (*domain*) of every variable is finite. The domain of each variable is saved and updated during the progression of calculations by using the constraints that involve this variable and others whose domain has been modified. These constraints take part in the elimination of all the inconsistent values of a variable from its domain; this is done by using some techniques called *filtering algorithms*. This means that all infeasible solutions are removed and only feasible solutions are effectively considered.

This method was applied for the physician scheduling problem by Cangini [11], Rousseau et al. [27], Trilling [32] and Bourdais et al. [6]. The work of Rousseau et al. [27] is about using constraint programming to define a general algorithm that takes into account two types of generic constraints: pattern and distribution constraints. We will not give more details about this general method, the interested reader is referred to [27].

This algorithm was successfully applied to two hospitals: SCH and CNH. The physician scheduling problem that is solved in [27] is formulated as follows:

Minimize  $f(W)$

subject to  $W_{ds} \in A_{ds}$

Distribution constraints

Pattern constraints.

The set  $A_{ds}$  contains the physicians who can work shift  $s$  of day  $d$ . The variable  $W_{ds}$  represents the physician who will be on duty on shift  $s$  of day  $d$ . As for the methods presented earlier in this section, the formulation of objective function  $f$  is the most difficult part of the solution scheme. In this case,  $f(W)$  represents the ‘cost’ associated to the schedules that are generated for all physicians (one schedule for each physician). The cost of the schedule for a given physician  $p$  is the sum of the penalties associated with each constraint.

## 5 Physician Scheduling and Nurse Rostering

As we mentioned earlier, there are several similarities between the physician scheduling problem that we consider in this paper and the nurse scheduling (or rostering) problem that is typically encountered in the wards of hospitals around the world (see Burke et al. [8] for a comprehensive state of the art of nurse rostering). In many ways, the constraints that one faces in both cases are of similar nature: for instance, Burke et al. classify the constraints tackled in nurse rostering into two broad categories, *coverage constraints*, which correspond to our demand constraints, and *time related constraints*; the various time related constraints that they list (capacity, personal preferences, consecutiveness, workload



balance, and others) include all of our other constraint types (fairness, ergonomics and personal preferences are no less important for nurses than they are for physicians). There is one area, however, where physician scheduling and nurse rostering problems do differ significantly and this is in the specification of the objective function to be optimized. As reported by Burke et al., minimization of personnel costs or of cost-related criteria is common in nurse rostering, while we never had to deal with cost issues in our case studies.

The nature of the solution techniques that have been applied for physician scheduling and nurse rostering is also pretty similar, except for the fact that Burke et al. report significant application of metaheuristic approaches other than tabu search (in particular, simulated annealing and genetic algorithms), as well as of expert and knowledge based systems (e.g., case-based reasoning approaches). Clearly, given the similarities between the two problems, there is absolutely no reason to believe that any given approach that works for one of them would not work for the other one. However, one should be careful about drawing hasty conclusions about the performance of specific algorithms or software. The main reason is that while the general structure of physician scheduling and of nurse rostering problems might be similar, one can expect specific instances to differ significantly. This is especially true with respect to demand/coverage constraints: as we pointed out earlier, in most physician scheduling problems, the demand for any given shift is usually one, while this value will typically be significantly larger in nurse rostering problems. The inclusion of cost-related criteria in the objective may also be expected to have a definite impact on the performance of some solution methods.

## 6 Conclusion

The physician scheduling problem is a challenging one. While we have proposed a series of generic constraints to describe it, it must be understood that the specific constraints that are in force in any given case study may vary wildly. This makes it difficult to come up with solution methods that can be used in a wide range of practical settings. It also greatly complicates the task of coming up with fair comparisons of different methods, since they may have been developed for settings that are quite different in nature. Indeed, we have attempted to compare the four approaches described in the previous section and found that just creating a set of benchmark instances that would allow such a comparison was in itself a very challenging task. We hope to be able to report on this comparison at a later date.

## References

1. Acme-Express: Medical staff and physician scheduling software (2000), <http://www.docs2000.net/productdetailsy2k.asp>
2. Beaulieu, H.: Planification de l'horaire des médecins dans une salle d'urgence. Master's Thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada (1998)

3. Beaulieu, H., Ferland, J.A., Gendron, B., Michelon, P.: A mathematical programming approach for scheduling physicians in the emergency room. *Health Care Management Science* 3, 139–200 (2000)
4. Berrada, I.: Planification d'horaires du personnel infirmier dans un établissement hospitalier. Ph.D. Dissertation, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada (1993)
5. Berrada, I., Ferland, J.A., Michelon, P.: A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences* 30, 183–193 (1996)
6. Bourdais, S., Galinier, P., Pesant, G.: HIBISCUS: A constraint programming application to staff scheduling in health care. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 153–167. Springer, Heidelberg (2003)
7. Burke, E.K., De Causmaecker, P., Vanden Berghe, G.: A hybrid tabu search algorithm for the nurse rostering problem. In: McKay, B., Yao, X., Newton, C.S., Kim, J.-H., Furuhashi, T. (eds.) SEAL 1998. LNCS (LNAI), vol. 1585, pp. 187–194. Springer, Heidelberg (1999)
8. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The state of the art of nurse rostering. *Journal of Scheduling* 7, 441–499 (2004)
9. Buzon, I.: La confection des horaires de travail des médecins dans une salle d'urgence résolue à l'aide de la méthode tabou. Master's Thesis, École Polytechnique, Montréal, Canada (2001)
10. ByteBloc Software: Epsked 3.0 bytebloc software (1995), <http://www.bytebloc.com>
11. Cangini, G.: A constraint programming local search algorithm for physician scheduling. Publication CRT-2000-26, Centre for Research on Transportation, Université de Montréal, Canada (2000)
12. Carter, M.W., Lapierre, S.D.: Scheduling emergency room physicians. *Health Care Management Science* 4, 347–360 (2001)
13. Chvátal, V.: *Linear Programming*. Freeman, New York (1983)
14. Dowland, K.A.: Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operation Research* 106, 393–407 (1998)
15. Forget, F.: Confection automatisée des horaires des médecins dans une salle d'urgence. Master's Thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Canada (2003)
16. Gendreau, M.: An introduction to tabu search. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 37–54. Kluwer Academic Publishers, Dordrecht (2003)
17. Gendreau, M., Hertz, A., Laporte, G.: A tabu search algorithm for the vehicle routing problem. *Management Science* 40, 1276–1290 (1994)
18. Gendreau, M., Soriano, P., Salvail, L.: Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research* 41, 385–403 (1993)
19. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 13, 533–549 (1986)
20. Knaunth, P.: The design of shift systems. *Ergonomics* 36, 15–28 (1993)
21. Knaunth, P.: Design better shift systems. *Applied Ergonomics* 27, 39–44 (1996)
22. Labbé, S.: La confection automatisée d'horaires pour les médecins en salles d'urgence. Master's Thesis, École des Hautes Études Commerciales de Montréal, Canada (1998)
23. Labit, P.: Amélioration d'une méthode de génération de colonnes pour la confection d'horaire d'infirmières. Master's Thesis, École Polytechnique, Montréal, Canada (2000)

24. Lloyd, S., Shannon, S., Steiner, D.: Burnout, depression, life and job satisfaction among Canadian emergency physicians. *Journal of Emergency Medicine* 12, 559–565 (1994)
25. Marriott, K., Stuckey, P.J.: *Programming with Constraints: An Introduction*. MIT Press, Cambridge, MA (1998)
26. Nemhauser, G.L., Wolsey, L.A.: *Integer and Combinatorial Optimization*. Wiley-Interscience, New York (1988)
27. Rousseau, L.M., Pesant, G., Gendreau, M.: A hybrid algorithm to solve a physician rostering problem. In: *Second Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* Paderborn, Germany (2000)
28. Skorin-Kapov, J.: Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing* 2, 33–45 (1990)
29. Taillard, É.: Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research* 47, 65–74 (1990)
30. Taillard, É.: Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17, 443–455 (1991)
31. Peake Software Laboratories: Tangier emergency physician scheduling software. <http://peakesoftware.com/peake/>
32. Trilling, G.: *Génération automatique d'horaires de médecins de garde pour l'hôpital Côte-des-Neiges de Montréal*. Publication CRT-98-05. Centre for Research on Transportation, Université de Montréal, Canada (1998)
33. Vovor, T.: *Problème de chemins bicritère ou avec contraintes de ressources: algorithmes et applications*. Ph.D. Thesis, École Polytechnique, Montréal (1997)
34. Weil, G., Heus, K., Poujade, P., François, M.: Constraint Programming for nurse scheduling. *Engineering in Medicine and Biology* 14, 417–422 (1995)
35. Warner, D.M.: Scheduling nursing personnel according to nursing preference: a mathematical programming approach. *Operations Research* 24, 842–856 (1976)

# A Flexible Model and a Hybrid Exact Method for Integrated Employee Timetabling and Production Scheduling

Christian Artigues<sup>1</sup>, Michel Gendreau<sup>2</sup>, and Louis-Martin Rousseau<sup>2</sup>

<sup>1</sup> Université de Toulouse, LAAS-CNRS, 7 avenue du Colonel Roche,  
31077 Toulouse, Cedex 4, France  
[artigues@laas.fr](mailto:artigues@laas.fr)

<sup>2</sup> CRT, Université de Montréal, CP 6128, succ. Centre-ville,  
Montreal, QC, Canada H3C 3J7  
[{michelg,louism}@crt.umontreal.ca](mailto:{michelg,louism}@crt.umontreal.ca)

**Abstract.** We propose a flexible model and several integer linear programming and constraint programming formulations for integrated employee timetabling and production scheduling problems. A hybrid constraint and linear programming exact method is designed to solve a basic integrated employee timetabling and job-shop scheduling problem for lexicographic minimization of makespan and labor costs. Preliminary computational experiments show the potential of hybrid methods.

## 1 Introduction

In production systems, the decisions related to scheduling jobs on the machines and the decisions related to employee timetabling are often made in a sequential process. The objective of job scheduling is to minimize the production costs whereas the objective of employee timetabling is to maximize employee satisfaction (or to minimize labor costs). Either the employee timetabling is first established and then the scheduling of jobs must take employee availability constraints into account or the scheduling of jobs is done first and the employees must then adapt to cover the machine loads. It is well known that optimizing efficiently an integrated process would both improve production costs and employee satisfaction. However, the resulting problem has generally been considered as too complex to be used in practical situations. Some attempts have been made [13,7,9,10,11,21] but mostly considering an oversimplified version of the employee timetabling problem. Nevertheless the integration of task scheduling and employee timetabling has been successfully developed in complex transportation systems [6,8,13,17,19,22,23]. In this paper we propose a model of integrated production and employee scheduling that takes account of the following possible specific characteristics of the production context:

- A) An employee that has started a task may be replaced at any moment by another employee (of the same skill) with no notable effect nor interruption of the processed task.

- B) An employee is not necessarily needed during all the processing time of a task but only at some time periods that can occur before, during and after the processed task (setups, removals, transportation).
- C) Because of the automated production process, or the nature of the tasks performed by the employee (e.g. supervision), an employee may perform several tasks simultaneously during a shift.
- D) The production process can be quasi-continuous (on a 24-hour basis) whereas the employee timetabling has to be discretized in periods (on an 8-hour basis for instance).
- E) The duration of a task may change depending on the number or on the skill of the assigned workers.

In Section 2, we review the related work dealing with the integration of task and employee scheduling and we give the position of the considered problem among the various production scheduling and employee timetabling problems. In Section 3, we propose different ILP formulations of the considered problem. A constraint programming formulation is proposed in Section 4. In Section 5, we propose a hybrid framework to solve the lexicographic minimization of makespan and labor costs. In Section 6, we provide the results of a preliminary computational experiment carried out on a set of employee timetabling and job-shop scheduling instances. Concluding remarks are drawn in Section 7.

## 2 Literature Review and Position of the Considered Problem

We review some of the integrated vehicle and crew scheduling methods in Section 2.1 and the previous work on integrated production scheduling and employee timetabling in Section 2.2. We give the position of the considered problem in Section 2.3.

### 2.1 Vehicle and Crew Scheduling

Integrated vehicle and crew scheduling is an active research area in transportation systems, see [6,8,13,17,19,22,23,28] among others.

We focus hereafter on some recent papers presenting different models and solution methods. Cordeau et al. [8] propose a Benders decomposition scheme to solve aircraft routing and crew scheduling problems. They use a set partitioning formulation for both the aircraft routing and the crew scheduling. In the first scheme, the primal subproblem involves only crew scheduling variables and the master problem involves only aircraft routing variables. Both the primal subproblem and master problem relaxation are solved by column generation. Integer solutions are found by a three-phase method, adding progressively the integrity constraints. More recently, Mercier et al. [23] have improved the robustness of the proposed model. Their method reverses the Benders decomposition proposed in [8] by considering the crew scheduling problem as the master problem.

Haase and Fridberg [19] propose a method to solve bus and driver scheduling problems. The problem is formulated as a set partitioning problem with additional constraints in which a column represents either a schedule for a crew or for a vehicle. The additional constraints are introduced to connect both schedule types. A branch-and-price-and-cut algorithm is proposed in which column generation is performed to generate both vehicle and crew schedules. The method is improved in [18] with a set partitioning formulation only for the driver scheduling problem that incorporates side constraints for the bus itineraries. These side constraints guarantee that a feasible vehicle schedule can be derived afterwards in polynomial time. Furthermore, the inclusion of vehicle costs in this extended crew scheduling formulation ensures the overall optimality of the proposed two-phase crew-first, vehicle-second approach.

Freling et al. [17] propose a method to solve bus and driver scheduling problems on individual bus lines. They propose a formulation that mixes the set partitioning formulation for crew scheduling and the assignment formulation for the vehicle scheduling problem. They compute lower bound and feasible solutions by combining Lagrangian relaxation and column generation. Columns correspond to crew scheduling variables. The constraints involving the current columns are relaxed in a Lagrangian way. The obtained Lagrangian dual problem is a single-depot vehicle scheduling problem (SDVSP). Once the Lagrangian relaxation is solved a new set of columns with negative reduced costs is generated. The method is iterated until the gap between the so-computed lower bound and an estimated lower bound is small enough. Feasible solutions are generated from the last feasible SDVSP and the current set of columns.

## 2.2 Production and Employee Scheduling

Specific employee scheduling problems involved in production scheduling are often tackled considering the job schedule is fixed. As a representative work in this area, Valls et al. [27] consider a fixed schedule in a multi-machine environment and consider the problem of finding the minimal number of workers. The problem is formulated as a restricted vertex coloring problem and a branch and bound algorithm is presented.

A large part of work involving both job scheduling and employee timetabling aims at keeping the number of required employees at each time period under a threshold, without considering the regulation constraints of employee schedules nor the individual preferences and skills of employees. Daniels and Mazzola [10] consider a flow-shop problem in which the duration of an operation depends on the selected mode to process an operation. Each mode defines a number of resources (workers) needed during the processing of the operation. The scheduling horizon is discretized in periods and at each time period, the number of workers cannot exceed a fixed number. Optimal and heuristics approaches are proposed. Daniels et al. [9] propose the same approach in a parallel machine context. Bailey et al. [3] and Alfares and Bailey [1] propose an integrated model and a heuristic for project task and manpower scheduling where the objective is to find a trade-off between labor cost and daily overhead project cost. The labor cost depends

on the number of employed workers at each time period. The daily overhead cost depends on the project duration. There are no machine constraints and the labor restrictions consist in setting a maximal number of workers per period. In [21], the authors propose a MILP to minimize the makespan in a flow-shop with multi-processor workstation as a primary objective and to determine the optimal number of workers assigned to each machine as a secondary objective. The sequence of jobs is fixed on each machine and the makespan is minimized through lot-streaming.

Faaland and Schmitt [16] consider an assembly shop with multiple workstations. Each task must be performed on a given workstation by a worker. There are production and late-delivery costs on one hand and labor cost linked to the total number of employees on the other hand. The authors study the benefits of cross-training which allows employees to have requisite skills for several work-centers. A heuristic based on a priority rule and on the shifting bottleneck procedure is proposed.

A more general problem (w.r.t. the timetabling problem) is studied by Daniels et al. [11]. They extend the model proposed in [10] to an individual representation of employees in a flow-shop environment. Each employee has the requisite skills for only a subset of machines and can be assigned to a single machine at each time period. The duration of a job operation depends on the number of employees assigned to its machine during its processing. The employees assigned to an operation are required during all its processing time. No schedule regulations are considered except unavailability periods. A branch and bound method is developed and the benefits of the level of worker flexibility for makespan minimization is studied.

In [20], Haït et al. propose a general model for integrating production scheduling and employee timetabling, based on the concepts of load center, configuration, employee assignment and sequence. A so-called load center is a subset of machines that can be managed simultaneously by a single employee. A configuration is a set of load centers defining a partition of a subset of machines. At each scheduling time period a single configuration is active. Hence, the number of load centers in a configuration gives the number of active employees. An employee assignment is an assignment of each load center of a configuration to a different employee. The authors define the configuration graph each node correspond to a possible configuration and there is an arc between two configurations that can be consecutive in time with a weight giving the cost of the configuration changeover. This model allows one to represent the simultaneous work of an employee on several machines. However, the computation method of the job durations performed simultaneously by the same operator is not provided. An example with two machines provided by the authors shows that the computation of this duration of a job amounts to solving a scheduling problem of the elementary tasks performed by the operator. Furthermore, it can happen in practice that more than one operator is needed during the processing of a job on a machine, which is not covered by the proposed model. In this model, a schedule is defined by the start time of the jobs and by a path (with possible

loops and cycles) in the graph of configurations with the employee assignment for each configuration of this path. The authors provide two examples of integrated resolution in a flow-shop context. In the first example, they propose a dynamic programming algorithm to find a feasible path in the configuration graph with a fixed number of equivalent operators and a fixed sequence of jobs. In the second example they propose a heuristic and a lower bound of the makespan in a flow-shop where the timetabling problem is reduced to the assignment of an employee to each machine, the duration of the jobs depending of the employee performance.

Drezet and Billaut [14] consider a project scheduling problem with human resources and time-dependent activities requirements. Furthermore, employees have different skills and the main legal constraints dictated by the workforce legislation have to be respected. The model is quite general. However, only human resources are considered since the considered context is not a production scheduling problem where machines are critical resources. A tabu search method is proposed as well as proactive scheduling techniques to deal with the uncertainty of the problem.

This brief summary of the state of the art reveals that, compared to the transportation domain, the integration of production scheduling and employee timetabling is in its earliest phase. Almost no existing approach tackles the complex regulation constraints of work nor the diversity of employee activities in modern production systems. Recently, more sophisticated models have been proposed but independently of the relevant literature in staff scheduling in other areas and without proposing a general solution methodology.

### 2.3 Position of the Considered Problem

There are several variants of the employee timetabling problem, see for instance the recent surveys [15,26]. In this paper we focus on only one of the problems presented in [12] called individual shift scheduling where each employee (or team of employees) is considered individually with its own skills and preferences. The time horizon is discretized in elementary time periods (shifts). At each period, a set of activities has to be performed and each activity requires a specific number of workers. The objective of the employee timetabling problem is to assign a single activity to each employee at each time period ('rest' activity included) in order to cover the demand for all activities. Such an assignment is called a schedule. There are restrictions on the possible schedules due to regulation constraints and employee profiles. The objective of the timetabling problem is to maximize the employee satisfaction.

There is also a large number of different production scheduling problems [25]. In this paper we consider a rather general problem where a set of jobs linked by precedence constraints has to be scheduled on a set of machines. Each job has a processing time, a release date, a due date and is assigned to a unique machine. A job cannot be interrupted once started and each machine can process at most one job simultaneously. The job scheduling problem lies in assigning a start time to each job with the objective to minimize the production costs.



We propose to integrate the two problems by associating to each job (processed on a machine) a set of activities (performed by the employees) such that assigning a start time to a job determines the period of each associated activity. From the employee timetabling point of view, the demand profile is not known in advance but is determined by the job schedule. From the job scheduling point of view, the possibility to start a job is subject to the presence of the employees able to perform the activities generated by this job. The employee profile is determined by the selected employee schedules. We will give several mathematical formulations of variants of this problem in Section 3.

### 3 ILP Models of Integrated Employee Timetabling and Machine Scheduling Problems

The model of integration proposed by [20] is centered on the concept of configuration which is a partition of the machines at a given time period such that each subset is managed by a single operator. In this paper we propose to perform the integration through the concept of *activity* which is widely used in the employee timetabling literature. We first provide a model with a common time representation for timetabling and scheduling (Section 3.1). Then we extend this model to the case where there is a time representation for employee timetabling and another time representation for job scheduling (Section 3.2). We show how these models can be extended to tackle the variability in job durations and machine assignment through the concept of modes (Section 3.3). The three latter models are based on time indexed and assignment variable formulations. In Section 3.4 we show how the set covering formulation usually used in efficient employee scheduling methods can also be used in the production scheduling context.

#### 3.1 Common Time Representation for Timetabling and Scheduling and Single-Mode Jobs

We consider the following employee timetabling and machine scheduling problem.

Let  $T$  denote a time horizon, discretized in a set of elementary time periods  $t = 0, \dots, T - 1$ . We consider an organization comprising a set of  $E$  employees  $\mathcal{E} = \{1, \dots, E\}$  and a set of  $m$  machines  $\mathcal{M} = \{1, \dots, m\}$ . There is set of  $A$  activities  $\mathcal{A} = \{1, \dots, A\}$  where each activity may be required by a job and has to be performed by one or several employees.  $\mathcal{A}_e$  is the set of activities employee  $e$  is able to perform.

The organization has to process a set of  $n$  jobs  $\mathcal{J} = \{1, \dots, n\}$  during the time horizon. Each job  $j$  has a known duration  $p_j > 0$  and requires for its execution a precise machine  $m_j$ . A binary matrix  $(b_{jk})_{1 \leq j \leq n, 1 \leq k \leq m}$  states if job  $j$  requires machine  $k$ , i.e.  $b_{jm_j} = 1$  and  $b_{jk} = 0, \forall k \neq m_j$ . A matrix  $(R_{ja})_{1 \leq j \leq n, 1 \leq a \leq A}$  is given where  $R_{ja}$  is the number of employees that have to perform activity  $a$  during the processing of job  $j$ . Each job  $j$  has a release date  $r_j$  and a due date  $d_j$ .

There are precedence constraints linking the jobs, represented by a directed graph  $G = (V, U)$  where  $V$  is the set of nodes including one node per job plus a dummy start node denoted 0 and a dummy end node denoted  $n + 1$ .  $U$  is the set of arcs representing the precedence constraints. Each arc  $(i, j)$  of  $U$  is valued by a (positive or negative) time lag  $d_{ij}$ .

There are also specific constraints on the activities that can be assigned to a given employee over time which will be described below. The objective of the considered employee timetabling and machine scheduling problem is to assign a start time to each activity and to assign exactly one activity to each employee at each time period.

We assume that there is a production cost  $W_{jt}$  if job  $j$  starts at time  $t$  and an employee satisfaction cost  $C_{eat}$  if employee  $e$  is assigned to activity  $a$  at time  $t$ .

$x_{jt}$  is a binary decision variable where  $x_{jt} = 1$  if job  $j$  starts at time  $t$  and  $x_{jt} = 0$  otherwise.  $y_{eat}$  is a binary decision variable such that  $y_{eat} = 1$  if employee  $e$  is assigned to activity  $a$  at time  $t$  and  $y_{eat} = 0$  otherwise. The problem can be formulated as follows:

$$\min \sum_{j=1}^n \sum_{t=0}^{T-1} W_{jt} x_{jt} + \sum_{e=1}^E \sum_{a=1}^A \sum_{t=0}^{T-1} C_{eat} y_{eat} \quad (1)$$

$$\sum_{t=0}^{T-1} x_{jt} = 1 \quad \forall j \in \mathcal{J} \quad (2)$$

$$x_{jt} = 0 \quad \forall j \in \mathcal{J}, \forall t \notin \{r_j, \dots, d_j - p_j\} \quad (3)$$

$$\sum_{j=1}^n \sum_{\tau=t-p_j+1}^t b_{jk} x_{j\tau} \leq 1 \quad \forall t \in \{0, \dots, T-1\}, \forall k \in \mathcal{M} \quad (4)$$

$$\sum_{t=0}^{T-1} t x_{jt} - \sum_{t=0}^{T-1} t x_{it} \geq d_{ij} \quad \forall (i, j) \in U \quad (5)$$

$$\sum_{e=1}^E y_{eat} \geq \sum_{j=1}^n \sum_{\tau=t-p_j+1}^t R_{ja} x_{j\tau} \quad \forall a \in \mathcal{A}, \forall t \in \{0, \dots, T-1\} \quad (6)$$

$$\sum_{a \in \mathcal{A}_e} y_{eat} = 1 \quad \forall e \in \mathcal{E}, \forall t \in \{0, \dots, T-1\} \quad (7)$$

$$Fy \leq f \quad (8)$$

$$x_{jt} \in \{0, 1\} \quad \forall j \in \mathcal{J}, \forall t \in \{0, \dots, T-1\} \quad (9)$$

$$y_{eat} \in \{0, 1\} \quad \forall e \in \mathcal{E}, \forall a \in \mathcal{A}, \forall t \in \{0, \dots, T-1\}. \quad (10)$$

The objective of the problem is to minimize the total cost (1) subject to the following constraints. Each job has to be started exactly once: (2). Each job must be started a way that it is started and finished within its time window: (3). At most one job can be processed by a machine at each time period: (4). The precedence

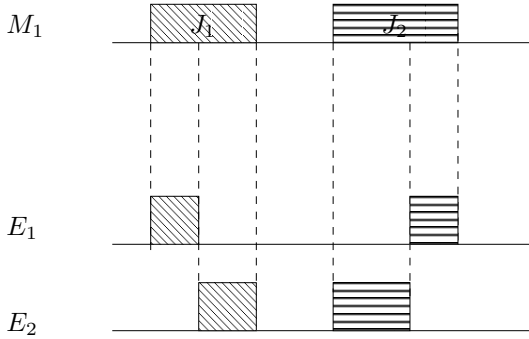


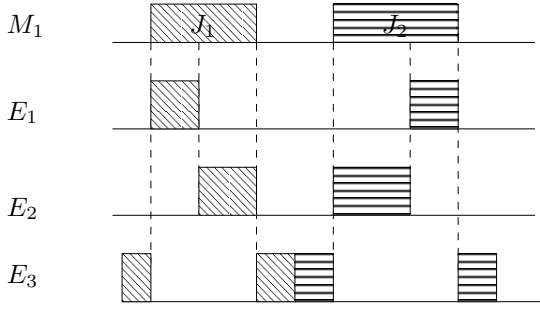
Fig. 1. A 1-machine and 2-employee example

constraint must be satisfied: (5). The number of employees assigned to each activity at each time period has to cover the total demand of all jobs in process: (6). Each employee has to be assigned to exactly one activity (in set  $\mathcal{A}_e$ ) at each time period: (7). We assume  $\mathcal{A}$  contains also non working activities representing employee inactivity (break, lunch, etc.) gathered in set  $\mathcal{P}$ . Constraints (8) are specific constraints for each employee  $e$  of the form  $\sum_{a \in \mathcal{A}} \sum_{t=0}^{T-1} F_{atq} y_{eat} \leq f_q$ , with  $F_{atq} \in \{-1, 0, 1\}$ , which allow for instance the taking into account of minimum or maximum consecutive periods of work, and other complex regulation constraints. For instance, if no employee can work more than two consecutive shifts, the constraints of the form  $\sum_{a \in \mathcal{A} \setminus \mathcal{P}} (y_{ea(t-1)} + y_{eat} + y_{ea(t+1)}) \leq 2$  can be defined for each time period  $t \in [1, T-2]$  for each employee  $e$ . The main drawback of this formulation is that the number of these constraints can be huge in practical situations and in general a set covering formulation is preferred (see Section 3.4).

The main difference between the machines and the employee resource is that employee timetables are more flexible, as illustrated in the example displayed in Figure 1. In this example, the two jobs generate a single activity during their processing. If we suppose that the first employee  $E_1$  allocated to this activity has to take a break while  $J_1$  is in process, another employee can perform the activity until the break of  $E_1$  is over which occurs in this example while  $J_2$  is in process.

### 3.2 Different Time Representations for Timetabling and Scheduling and Single-Mode Jobs

We assume that, for practical reasons, there may be a different time representation for the machine scheduling problem and for the employee timetabling problem. Let  $T$  denote the time horizon for the scheduling problem and let  $\Theta$  denote the time horizon for the timetabling problem. Furthermore, we assume



**Fig. 2.** A 1-machine and 3-employee example

that if a job  $j$  starts at time  $t$ ,  $0 \leq t < T$  then a number of employees  $R_{jat\theta} \geq 0$  is required to perform activity  $a$  at each period  $\theta$ ,  $0 \leq \theta < \Theta$ .

It follows that demand covering constraints (6) can be generalized with constraints (16) below and the new model is

$$\min \sum_{j=1}^n \sum_{t=0}^{T-1} W_{jt} x_{jt} + \sum_{e=1}^E \sum_{a=1}^A \sum_{\theta=0}^{\Theta-1} C_{ea\theta} y_{ea\theta} \quad (11)$$

$$\sum_{t=0}^{T-1} x_{jt} = 1 \quad \forall j \in \mathcal{J} \quad (12)$$

$$x_{jt} = 0 \quad \forall j \in \mathcal{J}, \forall t \notin \{r_j, \dots, d_j - p_j\} \quad (13)$$

$$\sum_{j=1}^n \sum_{\tau=t-p_j+1}^t b_{jk} x_{j\tau} \leq 1 \quad \forall t \in \{0, \dots, T-1\}, \forall k \in \mathcal{M} \quad (14)$$

$$\sum_{t=0}^{T-1} t x_{jt} - \sum_{t=0}^{T-1} t x_{it} \geq d_{ij} \quad \forall (i, j) \in U \quad (15)$$

$$\sum_{e=1}^E y_{ea\theta} \geq \sum_{j=1}^n \sum_{t=0}^{T-1} R_{jat\theta} x_{jt} \quad \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta-1\} \quad (16)$$

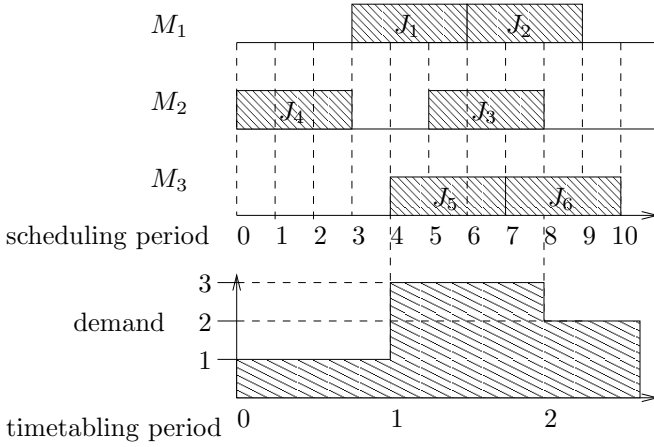
$$\sum_{a \in \mathcal{A}_e} y_{ea\theta} = 1 \quad \forall e \in \mathcal{E}, \forall \theta \in \{0, \dots, \Theta-1\} \quad (17)$$

$$Fy \leq f \quad (18)$$

$$x_{jt} \in \{0, 1\} \quad \forall j \in \mathcal{J}, \forall t \in \{0, \dots, T-1\} \quad (19)$$

$$y_{ea\theta} \in \{0, 1\} \quad \forall e \in \mathcal{E}, \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta-1\}. \quad (20)$$

Such constraints allow one to consider the cases where the employees need not be present during all the processing of a job on its machine, or when the employee activity generated by the job is not simultaneous with the processing



**Fig. 3.** A three-machine example with associated staffing demand

of the jobs. This feature takes place when employees have to perform setup or removal activities before and after the job processing, or when a control operation has to be carried out during a limited time while the job is in process. In Figure 2, a third employee is necessary only right before the start and right after the end of jobs  $J_1$  and  $J_2$ .

This type of model allows one also to take account of a different time scale between the time horizon of the scheduling problem, with the time periods considered in the timetabling problem. Suppose the scheduling time period is one hour and the timetabling period is four hours, then an aggregated information of the activities to perform during each four-hour period has to be provided. For this purpose, values  $R_{jat\theta}$  need not be integers if the activity  $a$  generated by job  $j$  in timetabling period  $\theta$  occupies only a portion of an employee’s work capacity. In Figure 3, each job is assumed to require 0.25 employees per time unit and generate a single activity. Then, the demand for employees able to perform this activity is displayed for each timetable period.

### 3.3 Multi-mode Jobs

We consider the case where for each job  $j$  there is a number  $Q_j$  of different processing modes corresponding to different ways (durations, machine and activity requirements) to perform job  $j$ . Let  $p_j^q$  denote the duration of job  $j$  in mode  $q$ . Let  $b_{jk}^q = 1$  if job  $j$  uses machine  $k$  in mode  $q$  and  $b_{jk}^q = 0$  otherwise.  $x_{jt}^q$  is a binary decision variable such that  $x_{jt}^q = 1$  if job  $j$  is started at time  $t$  in mode  $q$  and  $x_{jt}^q = 0$  otherwise.  $R_{jat\theta}^q$  now denotes the number of employees that must perform activity  $a$  at period  $\theta$  if job  $j$  is started at time  $t$  in mode  $q$ . The model can be adapted as follows:

$$\min \sum_{j=1}^n \sum_{q=1}^{Q_j} \sum_{t=0}^{T-1} W_{jt} x_{jt}^q + \sum_{e=1}^E \sum_{a=1}^A \sum_{\theta=0}^{\Theta-1} C_{ea\theta} y_{ea\theta} \quad (21)$$

$$\sum_{q=1}^{Q_j} \sum_{t=0}^{T-1} x_{jt}^q = 1 \quad \forall j \in \mathcal{J} \quad (22)$$

$$x_{jt}^q = 0 \quad \forall j \in \mathcal{J}, \forall q \in \{1, \dots, Q_j\} \\ \forall t \notin \{r_j, \dots, d_j - p_j\} \quad (23)$$

$$\sum_{j=1}^n \sum_{q=1}^{Q_j} \sum_{\tau=t-p_j^q+1}^t b_{jk}^q x_{j\tau}^q \leq 1 \quad \forall t \in \{0, \dots, T-1\}, \forall k \in \mathcal{M} \quad (24)$$

$$\sum_{q=1}^{Q_j} \sum_{t=0}^{T-1} t x_{jt}^q - \sum_{s=1}^{Q_i} \sum_{t=0}^{T-1} t x_{it}^s \geq d_{ij} \quad \forall (i, j) \in U \quad (25)$$

$$\sum_{e=1}^E y_{ea\theta} \geq \sum_{j=1}^n \sum_{q=1}^{Q_j} \sum_{t=0}^{T-1} R_{jat\theta} x_{jt}^q \quad \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta-1\} \quad (26)$$

$$\sum_{a \in \mathcal{A}_e} y_{ea\theta} = 1 \quad \forall e \in \mathcal{E}, \forall \theta \in \{0, \dots, \Theta-1\} \quad (27)$$

$$Fy \leq f \quad (28)$$

$$x_{jt}^q \in \{0, 1\} \quad \forall j \in \mathcal{J}, \forall q \in \{1, \dots, Q_j\}, \\ \forall t \in \{0, \dots, T-1\} \quad (29)$$

$$y_{ea\theta} \in \{0, 1\} \quad \forall e \in \mathcal{E}, \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta-1\}. \quad (30)$$

### 3.4 Set Covering Formulations

Let  $\mathcal{S}_e$  denote the set of valid schedules for an employee  $e$ . For each schedule  $s \in \mathcal{S}_e$ , each activity  $a$  and each timetabling period  $\theta$ , binary value  $y_{sa\theta}$  is such that  $y_{sa\theta} = 1$  if the schedule performs activity  $a$  at time  $\theta$  and  $y_{sa\theta} = 0$  otherwise.  $C_s$  denotes the cost of a schedule  $s \in \mathcal{S}_e$ . In addition, a binary decision variable  $z_s$  is defined such that  $z_s = 1$  if schedule  $s$  is selected and  $z_s = 0$  otherwise.

A new model can then be proposed by including the set covering formulation of the timetabling constraints (we ignore the multi-mode characteristics):

$$\min \sum_{j=1}^n \sum_{t=0}^{T-1} W_{jt} x_{jt} + \sum_{e=1}^E \sum_{s \in \mathcal{S}_e} C_s z_s \quad (31)$$

$$\sum_{t=0}^{T-1} x_{jt} = 1 \quad \forall j \in \mathcal{J} \quad (32)$$

$$\begin{aligned} x_{jt} &= 0 \quad \forall j \in \mathcal{J}, \\ &\quad \forall t \notin \{r_j, \dots, d_j - p_j\} \end{aligned} \quad (33)$$

$$\sum_{j=1}^n \sum_{\tau=t-p_j+1}^t b_{jk} x_{j\tau} \leq 1 \quad \forall t \in \{0, \dots, T-1\}, \forall k \in \mathcal{M} \quad (34)$$

$$\sum_{t=0}^{T-1} t x_{jt} - \sum_{t=0}^{T-1} t x_{it} \geq d_{ij} \quad \forall (i, j) \in U \quad (35)$$

$$\sum_{e=1}^E \sum_{s \in \mathcal{S}_e} y_{sa\theta} z_s \geq \sum_{j=1}^n \sum_{t=0}^{T-1} R_{jat\theta} x_{jt} \quad \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta-1\} \quad (36)$$

$$\sum_{s \in \mathcal{S}_e} z_s = 1 \quad \forall e \in \mathcal{E} \quad (37)$$

$$x_{jt} \in \{0, 1\} \quad \forall j \in \mathcal{J}, \forall t \in \{0, \dots, T-1\} \quad (38)$$

$$z_s \in \{0, 1\} \quad \forall e \in \mathcal{E}, \forall s \in \mathcal{S}_e. \quad (39)$$

## 4 A Constraint Programming Model

Constraint programming formulations have been proposed for production scheduling [4] and for employee timetabling [12]. We present hereafter an integrated formulation which involves start time decision variables  $S_j \in [r_i, d_i - p_i]$  for all jobs, an activity assignment variable  $a_{\theta e} \in \mathcal{A}_e$  giving the activity assigned to employee  $e$  in period  $\theta$  and a demand variable  $\delta_{\theta a} \in \mathcal{N}$  giving the number of employees required for activity  $a$  during period  $\theta$ . Consider the following constraint satisfaction problem (CSP):

$$S_j - S_i \geq d_{ij} \quad \forall (i, j) \in U \quad (40)$$

$$S_j + p_j \leq S_i \vee S_i + p_i \leq S_j \quad \forall i, j \in \mathcal{J}, m_i = m_j \quad (41)$$

$$\phi(\delta_{\theta a}, S) \quad \forall \theta \in \{0, \dots, \Theta-1\}, \forall a \in \mathcal{A} \quad (42)$$

$$\mathbf{distribute}((\delta_{\theta a})_{a \in \mathcal{A}}, \mathcal{A}, (a_{\theta e})_{e \in \mathcal{E}}) \quad \forall \theta \in \{0, \dots, \Theta-1\} \quad (43)$$

$$\mathbf{regular}((a_{\theta e})_{\theta \in \{0, \dots, \Theta-1\}}, \Pi) \quad \forall e \in \mathcal{E}. \quad (44)$$

Constraints (40) are the precedence constraints. Constraints (41) are the machine disjunctive constraints. Constraints (42) establish the link between the job start time variables  $S$  and the demand variable  $\delta$  through generic constraint  $\phi$  that needs to be specified for each specific problem. Constraints (43) represent demand satisfaction through the global cardinality constraint **distribute** which states that for a given period  $\theta$ ,  $\delta_{\theta a}$  variables must have value  $a$  in the activity assignment vector  $(a_{\theta e})_{e \in \mathcal{E}}$  of employees during period  $\theta$ . Last, constraints (44) express the employee specific and regulation constraints through the global regular language membership constraints **regular** [24], restricting the sequence

of values taken by the assignment variables to belong to the regular language associated to  $\Pi$ .

The advantage of constraint programming is its high flexibility to model complex demand computations, as well as complex regulation constraints.

The above CSP can be transformed into an optimization problem by introducing cost variables. This can be done through the `element` global constraints (see next Section). As an alternative, in [12], a new global constraint `cost - regular(X, Π, z, C)` extends the `regular` constraint by computing the cost  $z$  associated by an assignment of variables  $X$  given cost matrix  $C$ .

## 5 Solving a Lexicographic Makespan and Employee Cost Optimization Problem by a Hybrid LP-CP Method

In this section, we propose a hybrid CP-LP exact method to solve a lexicographic bicriteria optimization problem. The considered production cost is the makespan, denoted  $C_{\max}$ . Let  $C_{\text{empl}}$  denote the total satisfaction cost of employees. The considered problem can be denoted

$$\min Lex(C_{\max}, C_{\text{empl}}) \tag{45}$$

$$C_{\max} \geq S_j + p_j \quad \forall j \in \mathcal{J} \tag{46}$$

$$C_{\text{empl}} = \sum_{e \in \mathcal{E}} \sum_{\theta=0}^{\Theta-1} C_{e\theta} \tag{47}$$

$$\text{element}(C_{e\theta}, (C_{ea\theta})_{a \in \mathcal{A}_e}, a_{\theta e}) \quad \forall e \in \mathcal{E}, \forall \theta \in \{0, \dots, \Theta - 1\} \tag{48}$$

(40) ... (44).

Constraints (46) enforce the makespan value. Constraint (47) defines the total cost  $C_{\text{empl}}$  as the sum of elementary employee/period costs represented by decision variables  $C_{e\theta}$ . `element` global constraints (48) simply enforce the implications  $a_{\theta e} = v \implies C_{e\theta} = C_{ev\theta}$  for all  $\theta \in \{0, \dots, \Theta - 1\}$ ,  $e \in \mathcal{E}$  and  $v \in \mathcal{A}_e$ . The problem can be solved by first finding the optimal makespan  $C_{\max}^*$  (problem A) and, second, by finding the minimal employee cost  $C_{\text{empl}}^*$  compatible with  $C_{\max}^*$  (problem B).

We propose to solve both problems A and B through implicit enumeration in a constraint programming framework. Hence  $C_{\max}^*$  is found by iteratively searching the smallest  $V$  such that there is a feasible solution verifying  $C_{\max} \leq V$  (problem A).  $C_{\text{empl}}^*$  is found by searching the smallest  $V'$  such that there is a feasible solution verifying  $C_{\max} = C_{\max}^*$  and  $C_{\text{empl}} \leq V'$  (problem B).

At each node of each above-defined search trees, constraint propagation algorithms are performed to either reduce the domain of start time  $S$  and activity variables  $a$  or to detect an inconsistency and prune the node. The branching scheme first assigns values to start time variables and, once all start time variables are assigned, makes the remaining decisions on activity variables. Note that constraints  $\phi$  (42) have to ensure that once a complete assignment of the



start time variables is computed, the demand variables  $\delta$  are also completely assigned.

For both problems A and B, the makespan constraints set due dates on the job operations. Hence, standard scheduling constraint propagation algorithms can be used to reduce the start time domains. In the present work, we use precedence constraint propagation and edge-finding. We refer to [5] for a precise description of those algorithms.

For domain reduction of the demand and activity variables  $\delta$  and  $a$ , besides the standard **distribute** and **regular** constraint propagation algorithm, we propose to embed the linear programming relaxation of the ILP formulation (21) ... (30), limited to constraints involving  $y_{ea\theta}$  assignment variables, into a global constraint. Let  $\underline{\delta}_{\theta a}$  denote the smallest value in the domain of demand variable  $\delta_{\theta a}$  for activity  $a$  during period  $\theta$  at a given node of the constraint programming search tree. Then we consider the following LP relaxation, considering only labor costs:

$$\min \sum_{e=1}^E \sum_{a=1}^A \sum_{\theta=0}^{\Theta-1} C_{ea\theta} y_{ea\theta} \quad (49)$$

$$\sum_{e=1}^E y_{ea\theta} \geq \underline{\delta}_{\theta a} \quad \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta - 1\} \quad (50)$$

$$\sum_{a \in \mathcal{A}_e} y_{ea\theta} = 1 \quad \forall e \in \mathcal{E}, \forall \theta \in \{0, \dots, \Theta - 1\} \quad (51)$$

$$Fy \leq f \quad (52)$$

$$0 \leq y_{ea\theta} \leq 1 \quad \forall e \in \mathcal{E}, \forall a \in \mathcal{A}, \forall \theta \in \{0, \dots, \Theta - 1\}. \quad (53)$$

At a given node, the relaxation is stronger if the lower bound  $\underline{\delta}_{\theta a}$  on the demand is tight. This obviously depends on the definition and propagation of constraint  $\phi$ . Each time the LP relaxation is unfeasible, which can occur due to both demand undercoverage or labor cost upper bound violation, the current node is pruned.

Last, whenever an upper bound  $Z$  on the total labor cost  $C_{\text{empl}}$  is known, the reduced cost based filtering technique can be applied. Let  $\tilde{C}_{ea\theta}$  denote the reduced cost of an activity assignment variable  $y_{ea\theta}$  and let  $\underline{C}_{\text{empl}}$  denote the current optimal LP solution value. If,  $\underline{C}_{\text{empl}} + \tilde{C}_{ea\theta} \geq Z$ ,  $a$  can be removed from the domain of  $a_{\theta e}$ .

## 6 Computational Results on a Basic Employee Timetabling and Job-Shop Scheduling Problem

In this section, we show the potential of hybrid methods to solve integrated employee timetabling and production scheduling problems, through the resolution of basic employee and job-shop scheduling instances. For constraint based scheduling we use ILOG Solver 6.1 and Scheduler 6.1. For LP resolution we use

ILOG Cplex 9.1. All programs are coded in C++ under Linux on a AMD x86-64 architecture.

We consider the standard job-shop scheduling problem in which a job is made of  $m$  operations which form a chain in the precedence graph. Each job has to be processed by all the machines successively. Hence the operations of the same jobs are all assigned to different machines.

We consider job-shop instances of six jobs and four machines, comprising 24 operations. We consider a set of 15 employees and a set of 4 + 1 activities. The integer job operations processing times have been generated uniformly randomly between 1 and 10. We assume one time unit corresponds to one hour. We define a timetabling period as a eight-hour shift (i.e.  $T = 8\Theta$ ). Each employee has to be assigned to one activity during each shift. We assume activity 5 corresponds to employee inactivity during the shift. Each employee has skills for two production activities out of four. Each break must be of at least two consecutive shifts (16-hour break). There is an integer cost (uniformly randomly generated from 1 to 5) for assigning a production activity to an employee during each shift. Furthermore, to ensure problem feasibility at minimal makespan, we consider an additional set of 10 extra employees having a greater assignment cost (equal to 9 for all extra employees and for all periods and all activities).

We now describe how constraint  $\phi$  is implemented for the considered example. We simply assume there is a mapping between activities and machines. Hence, whenever a machine is in process during a shift, then an employee able to perform the corresponding activity is needed. It follows that at most four employees can be required simultaneously during a shift.

More precisely the link between the operation schedule  $S$  and the demand ( $\delta_{\theta a}$ ) can be described by the following constraints. Let  $D = T/\Theta$  and let  $\mathcal{J}_k$  denote the set of operations scheduled on machine  $k$ . Let  $a_k$  denote the activity corresponding to machine  $k$ :

$$\begin{aligned} S_j + p_j > D\theta \wedge S_j < D(\theta + 1) &\implies \delta_{\theta a_{m_j}} = 1 \\ \forall j \in \mathcal{J}, \forall \theta \in [0, \Theta - 1] \\ (S_j + p_j \leq D\theta \vee S_j \geq D(\theta + 1), \forall j \in \mathcal{J}_k) &\implies \delta_{\theta a_k} = 0 \\ \forall k \in \mathcal{M}, \forall \theta \in [0, \Theta - 1]. \end{aligned}$$

We use the standard job-shop resolution method provided in the example library of ILOG scheduler for the scheduling constraint propagation parts. For the search part on the start time and activity variables, we use a simple backtracking on possible values (in a chronological way for the start times). All employee constraints have been coded by `distribute` constraints. The LP relaxation and the reduced cost-based filtering algorithms are embedded into a global constraint. These algorithms are called whenever the lower bound of an activity demand is increased for any period or when the domain of a variable ( $a_{\theta e}$ ) is changed.

We have generated 10 instances having the above described characteristics. The results, comparing the hybrid method with and without reduced cost-based filtering, are displayed in Table [II](#). Column Inst gives the instance number.

**Table 1.** Method comparison on 10 basic employee and job-shop scheduling instances

Inst	Mks*	cost(M)	CPU(M)	#fails(M)	cost*	CPU(H)	#fails(H)	CPU(H <sup>-</sup> )	#fails(H <sup>-</sup> )
1	45	75	0.2s	3	29	0.8s	151	1.1s	438
2	56	69	0.2s	2	26	208s	27176	4099s	2459422
3	44	69	0.2s	2	26	2.2s	732	1.7s	1691
4	40	53	0.2s	3	23	0.5s	24	0.7s	183
5	40	63	0.2s	3	27	6.2s	4047	205s	117850
6	48	70	0.2s	7	28	0.9s	96	1.2s	371
7	43	67	0.2s	2	33	0.6s	83	0.8s	242
8	37	57	0.2s	3	22	28s	8185	400s	269799
9	49	69	0.2s	4	24(22)	3364s	340742	-	-
10	48	68	0.2s	3	23	4.1s	1140	408s	267695

Column Mks\* gives the optimal makespan obtained by pure CP without considering employee cost minimization. Column cost(M) gives the employee cost of the obtained solution. Columns #fails(M) and CPU(M) give the total number of fails and the CPU times of this search process. Column cost\* gives the minimal employee cost solution with a makespan equal to Mks\*. Columns #fails(H) and CPU(H) give the total number of fails and the CPU times of the complete hybrid search method needed to find the optimal cost solution. Columns #fails(H<sup>-</sup>) and CPU(H<sup>-</sup>) give the same values for the hybrid method used without reduced cost-based filtering.

For the proposed instances, the makespan minimization problem is very easy since CP always solves the problem in less than 0.2 s. Note that, in contrast, the hybrid methods outperform the standard constraint programming approaches for employee cost minimization since the latter is unable to find the optimal solution in a reasonable amount of time. Furthermore, while keeping the makespan optimal, the employee cost is significantly improved by the hybrid methods for all instances. One instance remains unsolved by all methods and the obtained lower and upper bounds are given as well as the total CPU time and number of fails needed to obtain them. This underlines the difficulty of the problem and shows the need for improvement of the proposed methods, considering also that the considered instances are small ones. The reduced cost-based filtering hybrid method outperforms the basic hybrid method on almost all instances showing the potential of high interaction between CP and LP for this kind of difficult integrated planning problem. In [2], enhanced hybrid methods and extended computational experiments are presented on the considered employee timetabling and job-shop scheduling problem.

## 7 Concluding Remarks

We have proposed a flexible model and several ILP and CP formulations for integrated employee timetabling and production scheduling. We have shown how the

flexibility of constraint programming modeling can be used to represent complex relationships between schedules and activity demands. A hybrid exact method involving standard constraint programming-based scheduling and timetabling technique on the one hand, and a linear programming relaxation with reduced-cost based filtering on the other hand, has been used to solve to optimality instances of the problem which cannot be solved by standard constraint programming. We are planning to generate several other instances to study the behavior of the proposed method with different problem characteristics. The search algorithm has also to be refined since we have used only standard backtracking schemes without any particular rule for activity selection. More realistic employee timetabling constraints will have also to be considered. This may lead to an improvement of the results of pure constraint programming techniques. The search could also be guided by using the linear programming optimal solution. Decomposition methods such as Benders decomposition or column generation will have also to be tested.

## References

1. Alfares, H., Bailey, J.: Integrated project task and manpower scheduling. *IIE Transactions* 29, 711–717 (1997)
2. Artigues, C., Gendreau, M., Rousseau, L., Vergnaud, A.: Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound. Technical Report 06700, Université de Toulouse, LAAS-CNRS, Toulouse, France (2006)
3. Bailey, J., Alfares, H., Lin, W.: Optimization and heuristic models to integrate project task and manpower scheduling. *Computers and Industrial Engineering* 29, 473–476 (1995)
4. Baptiste, P., Pape, C.L.: Disjunctive constraints for manufacturing scheduling: Principles and extensions. *International Journal of Computer Integrated Manufacturing* 9, 306–310 (1996)
5. Baptiste, P., Pape, C.L., Nuijten, W.: *Constraint-Based Scheduling*. Kluwer, Dordrecht (2001)
6. Bodin, L., Golden, B., Assad, A., Ball, M.: Routing and scheduling of vehicles and crews the state of the art. *Computers and Operations Research* 10, 63–211 (1983)
7. Chen, Z.: Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research* 129, 135–153 (2004)
8. Cordeau, J.F., Stojković, G., Soumis, F., Desrosiers, J.: Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Science* 35, 375–388 (2001)
9. Daniels, R.L., Hoopes, B.J., Mazzolla, J.B.: Scheduling parallel manufacturing cells with resource flexibility. *Management Science* 42, 1260–1276 (1996)
10. Daniels, R.L., Mazzolla, J.B.: Flow shop scheduling with resource flexibility. *Operations Research* 42, 504–522 (1994)
11. Daniels, R.L., Mazzolla, J.B., Shi, D.: Flow shop scheduling with partial resource flexibility. *Management Science* 50, 658–669 (2004)
12. Demasse, S., Pesant, G., Rousseau, L.M.: Constraint programming based column generation for employee timetabling. In: Barták, R., Milano, M. (eds.) *CPAIOR 2005*. LNCS, vol. 3524, Springer, Heidelberg (2005)

13. Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., Villeneuve, D.: A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In: Crainic, T., Laporte, G. (eds.) *Fleet management and logistics*, pp. 57–93. Kluwer, Dordrecht (1998)
14. Drezet, L.E., Billaut, J.C.: Tabu search algorithms for a predictive and a reactive project scheduling problem. In: *6th Metaheuristics International Conference*, Vienna (2005)
15. Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153, 3–27 (2004)
16. Faaland, B., Schmitt, T.: Cost-based scheduling of workers and equipment in a fabrication and assembly shop. *Operations Research* 41, 253–268 (1993)
17. Freling, R., Huisman, D., Wagelmans, A.: Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling* 6(1), 63–85 (2003)
18. Haase, K., Desaulniers, G., Desrosiers, J.: Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science* 35, 286–303 (2001)
19. Haase, K., Friberg, C.: An exact algorithm for the vehicle and crew scheduling problem. In: Wilson, N. (ed.) *Computer-Aided Transit Scheduling*. *Lecture Notes in Economics and Mathematical Systems*, vol. 471, pp. 63–80. Springer, Berlin (1999)
20. Haït, A., Baptiste, P., Brauner, N., Finke, G.: Approches intégrées à court terme. In: Baptiste, P., Giard, V., Haït, A. (eds.) *Gestion de production et ressources humaines*, ch. 6, *Presses Internationales Polytechnique* (2005)
21. Huq, F., Cutright, K., Martin, C.: Employee scheduling and makespan minimization in a flow shop with multi-processor work stations: a case study. *Omega* 32, 121–129 (2004)
22. Klabjan, D., Johnson, E., Nemhauser, G.: Airline crew scheduling with time windows and plane count constraints. *Transportation Science* 36, 337–348 (2002)
23. Mercier, A., Cordeau, J.F., Soumis, F.: A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers and Operations Research* 32, 1451–1476 (2005)
24. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) *CP 2004*. *LNCS*, vol. 3258, pp. 482–495. Springer, Heidelberg (2004)
25. Pinedo, M.L.: *Planning and Scheduling in Manufacturing and Services*. *Springer Series in Operations Research and Financial Engineering*, Springer, Berlin (2005)
26. Soumis, F., Pesant, G., Rousseau, L.M.: Gestion des horaires et affectation du personnel. In: Baptiste, P., Giard, V., Haït, A. (eds.) *Gestion de production et ressources humaines*, ch. 4, *Presses Internationales Polytechnique* (2005)
27. Valls, V., Pérez, A., Quintanilla, S.: A graph colouring model for assigning a heterogeneous workforce to a given schedule. *European Journal of Operational Research* 90, 285–302 (1996)
28. Voß, S., Daduna, J. (eds.): *Computer Aided Scheduling of Public Transport*. *Lecture Notes in Economics and Mathematical Systems*, vol. 505. Springer, Berlin (2001)

# Mememes, Self-generation and Nurse Rostering

Ender Özcan

Yeditepe University, Department of Computer Engineering,  
34755 Kadıköy/Istanbul, Turkey  
eożcan@cse.yeditepe.edu.tr

**Abstract.** This paper presents an empirical study on memetic algorithms in two parts. In the first part, the details of the memetic algorithm experiments with a set of well known benchmark functions are described. In the second part, a heuristic template is introduced for solving timetabling problems. Two adaptive heuristics that utilize a set of constraint-based hill climbers in a co-operative manner are designed based on this template. A hyper-heuristic is a mechanism used for managing a set of low-level heuristics. At each step, an appropriate heuristic is chosen and applied to a candidate solution. Both adaptive heuristics can be considered as hyper-heuristics. Memetic algorithms employing each hyper-heuristic separately as a single hill climber are experimented on a set of randomly generated nurse rostering problem instances. Moreover, the standard genetic algorithm and two self-generating multimeme memetic algorithms are compared to the proposed memetic algorithms and a previous study.

## 1 Introduction

Genetic Algorithms (GAs), as presented by J. Holland in [28], are very promising for tackling complex problems [24]. There are some shortcomings of generic genetic algorithms, such as premature convergence. The effectiveness of the use of appropriate operators and hybrid approaches is underlined by many researchers to overcome such difficulties. Memetic Algorithms (MAs) embody a class of algorithms that combine genetic algorithms and hill climbing methods [15, 38, 45, 46]. A *meme* represents a hill climbing method to be used within an MA as the local refinement component. Ning et al. [39] concluded from their experiments that the meme choice in an MA influences the performance significantly. Krasnogor [31] extended his previous studies and suggested a *self-generating multimeme* MA for solving problems in the existence of multiple choices for the operators. Each meme encodes an operator and its parameters that a candidate solution will employ during the evolution. In this study, a meme denotes a hill climbing method and its related parameters. Each meme is co-evolved with each candidate solution. The evolutionary process offers a learning mechanism to fully utilize the provided hill climbers [32, 34].

In the first part of this study, the multimeme approach proposed by Krasnogor [33] is tested on a set of benchmark functions. The study aims to answer the following questions:

- Can the suggested learning mechanism discover useful hill climbers?
- Does a set of hill climbers generate a synergy to obtain the optimal solution?

In the second part of this study, MAs for solving a nurse rostering problem, introduced by Özcan [41], are considered. Özcan extended the study by Alkan and Özcan [5], and suggested templates designing a set of operators, including a self-adjusting violation-directed and constraint-based heuristics, named as VDHC, within MAs for solving timetabling problems. A new heuristic template for managing a set of constraint-based hill climbers is introduced in this paper. Two new instances based on this template are implemented and used as a single hill climber within MAs. Furthermore, two multimeme memetic algorithms (MMAs) are described. The performances of all the proposed algorithms, including the traditional genetic algorithm and the MA provided in [41], are compared. A *hyper-heuristic* is a mechanism used to decide which heuristic to apply from a set of heuristics to a given candidate solution. It is an emerging search and optimization tool [11, 14, 43]. A variety of approaches are used as hyper-heuristics, including meta-heuristics. Several heuristics presented in this study for nurse rostering can be considered as hyper-heuristics used during the hill climbing process to manage a set of hill climbers within MAs.

In the following section, the nurse rostering problem and memetic algorithms, including the multimeme approach are summarized. The relevant details of MAs and the experiments for the benchmark function optimization are explained in Section 3. MAs, including the hill climbers, are described for solving the nurse rostering problem at a Turkish hospital in Section 4. The nurse rostering data and the experimental results are provided in Section 5. Finally, the conclusions are presented in Section 6.

## 2 Background

### 2.1 Nurse Rostering Problem

Timetabling problems are real-world constraint optimization problems. Due to their NP complete nature [21], traditional approaches might fail to generate a solution for an instance. Timetabling problems can be represented in terms of a three-tuple  $\langle V, D, C \rangle$ , where  $V$  is a finite set of *variables*,  $D$  is a finite set of *domains* of variables and  $C$  is a set of *constraints* to be satisfied:

$$V = \{v_1, v_2, \dots, v_M\}, \quad D = \{d_1, \dots, d_i, \dots, d_M\}, \quad C = \{c_1, c_2, \dots, c_K\}.$$

Solving a timetabling problem instance requires a search for finding the best assignment for all variables that satisfy all the constraints. Thus, a candidate solution is defined by an assignment of values from the domain to the variables:

$$V' = \{v_1 = v'_1, \dots, v_i = v'_i, \dots, v_M = v'_M\},$$

where  $v'_i \in d_i$  and  $d_i \subseteq D_1 \times \dots \times D_P$ , where  $P \geq 1$ .

In all timetabling problems, the time domain is a trivial domain for the variables. A problem instance might require other resources to be scheduled as well. For example, a university course timetabling problem instance might require the arrangement of the classrooms for each course meeting, as well. Then the search will be performed within a domain that will be a Cartesian product of time and classroom sets.

A *nurse roster* is a timetable consisting of employee shift assignments and the rest days of nurses in a health-care institution. Some health-care institutions might be composed of several departments. A *departmental roster* is defined as a collection of the nurse rosters of all nurses working within the same department. Nurse Rostering Problems (NRPs) are timetabling problems that seek for satisfactory schedules to be generated for the employees, employers and even for the customers. In a common NRP, a nurse can be assigned to a day or a night shift, or can stay off-duty. A variable represents the shift assignment of a nurse. In this paper, *event* and *daily shift* will be used to refer to *variable*, interchangeably. A *group* of events indicates a subset of events in  $V$  and their assignments in a candidate solution.

In all timetabling problems, constraints are classified as *hard* or *soft*. Hard constraints must be satisfied, while soft constraints represent preferences that are highly preferred. Furthermore, there are six different constraint categories for practical timetabling: *edge constraints*, *exclusions*, *presets*, *ordering constraints*, *event-spread constraint* and *attribute constraints* (includes *capacity constraints*) [22, 42]. Edge constraints are the most common constraints that represent pairs of variables to be scheduled without a clash. A timetabling problem reduces to a graph colouring problem if the instance requires only edge constraints to be satisfied [35]. Exclusions determine the members to be excluded from the domain of variables for each variable. Presets are used to fix the assignment of a variable. Ordering constraints, as the name suggests, are used to define an ordering between a pair of variables based on the timeline. Event-spread constraints define how the events will be spread out in time. Attribute constraints deal with the restrictions that apply between the attributes of a variable and/or the attributes of its assignment. Numerous researchers deal with NRPs based on different types of constraints utilizing variety of approaches. A recent survey on nurse rostering can be found in [10].

Burke et al. [8] applied variable neighbourhood search using a set of different perturbation methods and local search algorithms on randomly generated schedules. Chun et al. [13] modelled nurse rostering as a constraint satisfaction problem and embedded it as a Rostering Engine into the Staff Rostering System for the Hong Kong Hospital Authority. Similarly, Li et al. [36] modelled nurse rostering as a weighted constraint satisfaction problem. Their algorithm consists of two phases. In the first phase, forward checking, variable ordering and compulsory back-jumping are used, whereas in the second phase descend local search and tabu search are used. Ahmad et al. [2] proposed a population-less co-operative genetic algorithm and experimented on a three-shift problem. Kawanaka et al. [30] attempted to meet absolute and desirable constraints for



obtaining optimal nurse schedules. Aickelin et al. proposed a co-evolutionary pyramidal GA and experimented an indirect representation using three different decoders within GA for solving NRP in [3] and [4], respectively. Gendreau et al. [23] used TS to generate shifts of nurses at the Jewish General Hospital of Montreal. Berrada et al. [6] combined TS with multi-objective approach, prioritizing the objectives. A swap operator is used as a heuristic that swaps working and rest days in a roster. Duenas et al. [19] applied an interactive sequential multi-objective problem solving method in conjunction with a genetic algorithm to produce a weekly schedule of eight nurses. Burke et al. [7] compared steepest descent, traditional TS and its hybrid with two local search heuristics for solving nurse rostering problem in Belgian Hospitals.

Recently, research on timetabling started to move towards finding a good *hyper-heuristic* [11]; a heuristic for selecting a heuristic among a set of them to solve an optimization problem. Cowling et al. [14] described hyper-heuristics as an iterative search method which maintains a single candidate solution and a set of heuristics. A hyper-heuristic is a heuristic utilized to choose a lower level heuristics. Han et al. [29] compared different versions of hyper-heuristics based on a GA they developed for solving a trainer scheduling problem utilizing fourteen different lower level heuristics. Burke et al. [12] proposed a tabu-search based hyper-heuristic, demonstrating its success for solving a set of nurse rostering problems at a UK hospital.

## 2.2 Multimeme Algorithms

Memetic algorithms (MAs) are population-based hybrid algorithms that combine Genetic Algorithms and hill climbing [15, 38, 45, 46]. In MAs, a *chromosome* (*individual*) represents a candidate solution to a problem at hand. A *gene* is a subsection of a chromosome that encodes the value of a single parameter (*allele*). Generally, the search for an optimal solution starts with a randomly generated set of individuals, called *initial population*. Then at each evolutionary step (*generation*) a set of operators are applied to each individual in the population. First, *mates* are selected for performing *crossover*, an operator that exchanges genetic material between mates. While selecting the mates, better ones are preferred. For example, *tournament selection* returns the individual having the best fitness value as a mate among a set of randomly selected individuals of size *tour*. After the crossover, a set of new individuals, called *offspring*, is generated. Offspring are then *mutated*. In MAs, a hill climbing operator is applied to the individuals, right after the crossover, or the mutation or in both places. Even the initial generation can be hill climbed. Finally, the individuals in the current population are replaced by the offspring forming the next generation using a *trans-generational* strategy. Whenever the *termination criteria* are satisfied, the evolution stops. The best individual in the last generation is the best candidate solution achieved. In this paper, all MAs utilize a hill climber after the initialization and mutation.

Using a set of hill climbers, different MAs can be generated and compared for solving a problem. As another possibility, all hill climbers can be combined under a heuristic that selects one hill climber at a time and applies it. Such a

hyper-heuristic schedules a hill climber in a *deterministic* or a *non-deterministic* way. For example, a deterministic round-robin strategy schedules the next hill climber in a queue. A non-deterministic strategy schedules the next hill climber randomly. These approaches employ blind choices. More complex and smart hyper-heuristics can be designed by making use of a learning mechanism that gets a feedback from the previous choices to select the right hill climber at each step. Different types of hyper-heuristic are presented in [11]. Özcan et al. discuss four different hyper-heuristic frameworks in [43] for utilizing mutational and hill climbing heuristics simultaneously.

Multimeme algorithms (MMAs) represent a subset of self-generating (co-evolving) MAs [31–34]. An individual in a population carries memetic material along with a genetic material. A meme within memetic material indicates an operator and its relevant settings. The materials are co-evolved. In an evolutionary cycle, the memes are inherited to the offspring from the parents using the Simple Inheritance Mechanism (SIM) [33] during the crossover. SIM favours the meme of a mate with a better fitness to be transmitted to the offspring. In the case of an equal quality, a meme is randomly selected from the mates. Furthermore, a meme is altered to a random value based on a probability, called *Innovation Rate* (IR), during the mutation. MMAs, based on the SIM strategy and the mutation, allow modification of the candidate solutions by learning in order to obtain improved ones. This mechanism is referred to as the Lamarckian learning mechanism [31, 40].

Using a similar notation as provided in [33], a *meme*, denoted by  $HhAbInRt$ , represents the hill climbing method (H), its acceptance strategy (A), the maximum number of iterations (I), and which part of the configuration to apply the selected method (R). An individual uses its meme to decide the hill climbing method and the related components to use, after the mutation takes place. Previously, Ong et al. [40] conducted tests on three benchmark functions using two new methods that they proposed for selecting the appropriate meme within MAs. In this study, MMAs are extensively tested on a set of well known benchmark functions. Furthermore, MMAs are used to determine where to apply a hill climber and which hill climber to apply, self-adaptively, for solving a real-world nurse rostering problem.

*Success rate* (*s.r.*) indicates the ratio of successful runs, achieving the expected fitness to the total number of runs repeated. Comparisons of MAs are based on the average number of evaluations and the success rate. Additionally, *average evolutionary activity* is considered during the assessment of MMA experiments. *Evolutionary activity* of a meme at a given generation is the total number of appearance of itself within each population starting from the initial generation until the given generation. Average evolutionary activity is obtained by taking an average of the evolutionary activity of a meme at each generation over the runs. The slope of the average evolutionary activity versus generation curve shows how much a meme is favoured. The steeper the slope gets for a meme, the more it is favoured.

**Table 1.** Benchmark functions used during the experiments: *lb* and *ub* indicate the lower and upper bound for each dimension, respectively, *opt* indicates the optimum

Label	Function name	lb	ub	opt	Source
F1	Sphere	-5.12	5.12	0	[17]
F2	Rosenbrock	-2.048	2.048	0	[17]
F3	Step	-5.12	5.12	0	[17]
F4	Quartic <i>with noise</i>	-1.28	1.28	1	[53, 17]
F5	Foxhole	-65.536	65.536	1	[17]
F6	Rastrigin	-5.12	5.12	0	[47]
F7	Schwefel	-500	500	0	[50]
F8	Griewangk	-600	600	0	[27]
F9	Ackley	-32.768	32.768	0	[1]
F10	Easom	-100	100	-1	[20]
F11	Schwefel's Double Sum	-65.536	65.536	0	[51]
F12	Royal Road	-	-	0	[37]
F13	Goldberg	-	-	0	[25, 26]
F14	Whitley	-	-	0	[54]

### 3 Memetic Algorithms for Benchmarking

#### 3.1 Benchmark Functions and Hill Climbing Methods

Benchmark functions with different features, well known among the evolutionary algorithm researchers, are utilized during the experiments (Table 1). F1–F11 are continuous, whereas F12–F14 are discrete benchmark functions. Detailed properties of each function can be found in the source references presented in Table 1. Benchmark functions include De Jong's test suite [17]. The only difference is that the noise component of the Quartic function is modified as described in [53].

Eight hill climbers (memes) are used in the experiments:

- *Steepest Descent Hill Climber*. (MA0) [37] evaluates all states obtained by flipping each bit in a binary string and accepts the one with the lowest fitness.
- *Next Descent Hill Climber*. (MA1) [37] flips the bit in question and accepts the change if there is a fitness improvement at each step. This process is repeated starting from the least significant bit towards the most significant one.
- *Random Mutation Hill Climbing*. (MA2) [37] flips a randomly selected bit and accepts the change if there is a fitness improvement at each step.
- *Davis's Bit Hill Climbing*. (MA3) [16] employs the same improvement process as in MA1 at each step to each bit in a random order.

The remaining four memes are derived from the first two memes. The bit flip operation in MA0 and MA1 is replaced by an AND operation with 0, yielding MA4 and MA6, respectively. Similarly, an OR operation with 1 is employed, yielding MA5 and MA7, respectively. Gray and binary encodings are used to

**Table 2.** Common parameter settings used during the benchmark function experiments

Label	Dim.	No. of bits	Chrom. length	Pop. size	Max. hc steps
F1	10	30	300	60	600
F2	10	30	300	60	600
F3	10	30	300	60	600
F4	10	30	300	60	600
F5	2	30	60	20	120
F6	10	30	300	60	600
F7	10	30	300	60	600
F8	10	30	300	60	600
F9	10	30	300	60	600
F10	6	30	180	36	360
F11	10	30	300	60	600
F12	8	8	64	20	128
F13	30	3	90	20	180
F14	6	4	24	20	48

represent candidate solutions during benchmark experiments for continuous and discrete functions, respectively. The Gray encoding is preferred, since a bit flip generates a Hamming distance of one from the previous state and causes a small change in the decoded value. Due to the Gray encoding, the memes MA4-MA7 represent *poor* hill climbers for almost all continuous benchmark functions.

### 3.2 Experimental Setup

All runs are repeated 50 times. Pentium IV 2 GHz machines with 256 MB RAM are used during the experiments. The chromosome length,  $l$ , is the product of dimensions and the number of bits used. All the related parameters are arbitrarily chosen with respect to  $l$ . The mutation rate is chosen as a factor of  $1/l$ . The rest of the common parameter settings used during the experiments are presented in Table 2. Runs are terminated whenever the overall CPU time exceeds 600 s, or an expected fitness is achieved. All MAs use a tournament mate selection strategy with a tour size two, one point crossover, bit-flip mutation and a trans-generational MA with a replacement strategy that keeps only two best individuals from the previous generation. The IR rate is fixed as 0.20 during all multimeme experiments. A single acceptance strategy that approves only improving moves and a single value for the maximum number of hill climbing steps are used:  $b = \{1\}$  and  $n = \{l\}$ . A hill climber is applied to the whole individual:  $t = \{\text{whole}\}$ .

During the initial set of experiments, the benchmark functions are tested using each meme described in Section 2.1. Experiments are also performed using a traditional Genetic Algorithm for comparison. The second set of experiments is designed according to the results obtained from the initial one. The best meme and two poor memes are fed into a multimeme algorithm. In the last

set of MMA experiments, eight memes are used. Four hill climbing methods;  $h = \{MA0, MA1, MA2, MA3\}$  are embedded. Hill climbing is applied depending on the acceptance strategy,  $b = \{0, 1\}$ . 0 indicates a rejection, so hill climbing is not applied. If the meme points to the acceptance strategy 1, then the related hill climbing operator is applied. Hence, effectively there are five different memes. For short notation, each meme is referred to as GA, MA0–MA3.

### 3.3 Empirical Results for the Benchmark Functions

The performance comparisons of the genetic algorithm and the memetic algorithms using different memes are demonstrated in Figure 1 for selected benchmark functions. For each experiment, the related bar appears in the figure, only if all the runs yield the expected result. If two algorithms have a matching success, then the average number of evaluations that each algorithm requires is compared. An algorithm is considered to deliver a better performance, if the average number of evaluations is less than the other one. MA0 is the best meme choice for F4, F13 and F14. MA1 is the best meme choice for F6–F8. MA3 is the best meme choice for F2, F3, F5, F10, and F12. For functions F1, F9 and F11 genetic algorithm performs slightly better than the memetic algorithm with the meme MA1. MA2 and MA3 turn out to be the worst and the best meme, respectively, among MA0–MA3.

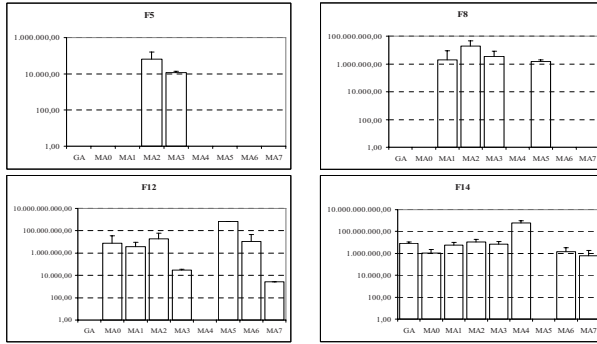
The average evolutionary activity versus generation plots generated during the second set of experiments show that the multimeme approach successfully identifies useful memes. The MMA chooses the best meme and applies it more than the rest of the memes for all benchmark function, as illustrated in Figure 2 for selected benchmark functions. The success rate for each benchmark function is 1.00. Any hill climber seems to attain the optimum fast for F1, F3 and F11.

In the third and the last set of experiments, results similar to the previous one are obtained. The MMA can still identify the best meme or a meme that does not perform significantly better than the best meme for almost each benchmark function, as shown in Figure 3 for selected benchmark functions. Furthermore, in all runs full success is achieved for all cases. Unfortunately, a synergy between hill climbers is not observed. Comparing the experimental results obtained using the MMA and the MA with the best meme for each benchmark indicates that the MA with the best meme is superior based on the average number of evaluations, except for F1, F3 and F11 (Table 3).

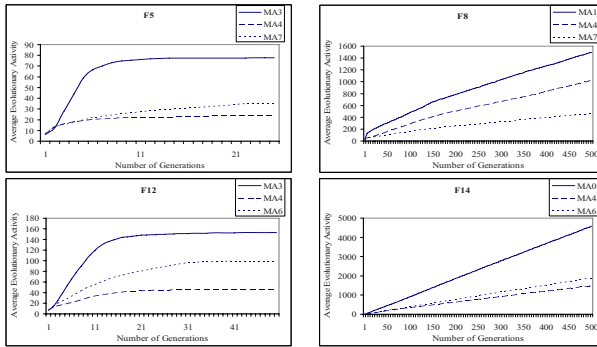
## 4 Memetic Algorithms for Nurse Rostering

### 4.1 Nurse Rostering Problem at a Turkish Hospital (NRPmh)

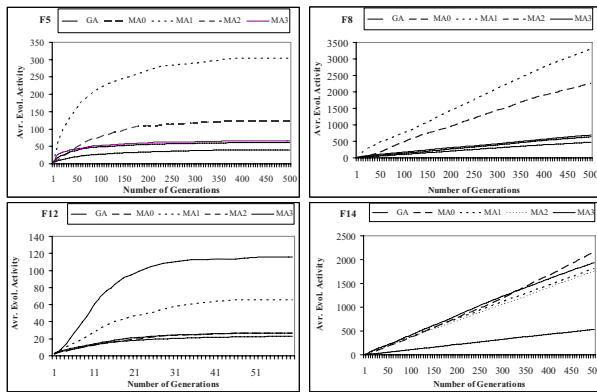
An analysis is performed on the Nurse Rostering Problem at a Turkish hospital in Istanbul, denoted as NRPmh. There are three types of daily shifts: *day*, *night* and *off-duty*. The timetable size is known in advance. Although a biweekly schedule is preferred, the hospital authorities produce a weekly schedule manually, in order



**Fig. 1.** Mean and the standard deviation of the number of evaluations per run, generated by each MA for a selected subset of benchmark functions



**Fig. 2.** Average evolutionary activity versus generation plots of each meme utilized during the second set of experiments for a selected subset of benchmark functions



**Fig. 3.** Average evolutionary activity versus generation plots of each meme utilized during the third set of experiments for a selected subset of benchmark functions

**Table 3.** Average number of evaluations and standard deviations generated by a memetic algorithm for each benchmark function: MA0-MA3 denotes the memetic algorithm using only the corresponding meme and MMA denotes the multimeme algorithm using all of them

Label	Type	Av. no evals	SD	Label	Type	Av. no evals	SD
F1	MMA	17,580	2,226	F8	MMA	5,215,787	9,658,230
	MA1	92,256	0		MA1	1,906,134	6,646,991
F2	MMA	23,605,004	24,364,979	F9	MMA	43,871	12,193
	MA3	8,455,507	3,803,504		MA1	180,783	12,647
F3	MMA	72,252	11,772	F10	MMA	3,100,515	4,565,736
	MA3	82,769	16,512		MA3	1,340,811	988,971
F4	MMA	12,926,879	11,435,876	F11	MMA	17,580	2,226
	MA0	9,494,844	10,332,574		MA1	36,060	0
F5	MMA	46,975	79,394	F12	MMA	31,297	14,961
	MA3	11,619	2,293		MA3	29,246	4,936
F6	MMA	553,306	231,124	F13	MMA	7,667,352	2,832,376
	MA1	525,398	262,055		MA0	4,348,896	1,617,951
F7	MMA	349,250	324,544	F14	MMA	3,674,932	2,623,300
	MA1	167,799	60,577		MA0	1,072,117	1,111,825

to simplify the timetabling process. Since the preferences of nurses are essential and might change in time, schedules are acyclic.

The hospital consists of three departments. Cross duty between the departments does not occur frequently. Hence, each nurse can be considered to be independent belonging to a specific department. Nurses are categorized into three ranks according to their experiences. Ranks  $\{0, 1, 2\}$  indicate the level of experience from lowest to highest. There are not many experienced nurses with rank 2, but there is at least one such nurse at each department. The constraints of this problem include the following.

Excludes:

- Exclude Night Shifts Constraint (ENC): Night shifts cannot be assigned to an experienced nurse with rank 2.

Event-spread constraints:

- Off-duty constraint (RDC): Nurses can define at most 4 rest day preferences.
- Shift constraint (SHC): At a department, during each shift there must be at least one nurse.
- Successive night shifts constraint (SNC): A nurse cannot be assigned to more than two successive night shifts.
- Successive day shifts constraint (SDC): A nurse cannot be assigned to more than three successive day shifts.
- Successive shifts constraint (SSC): A nurse cannot be assigned to two successive shifts. A day shift in one day and a night shift in the following day are considered as successive shifts.

```

1. while (termination criteria are not
   satisfied) do
   a. Select a group (or groups) of events based on violations
   b. Select a constraint type based on contribution of each
   constraint type within the selected group (or groups)
   c. Apply hill climbing for the selected constraint type
   (without considering the other constraints) within the
   selected group of events
2. end while

```

**Fig. 4.** Pseudo-code of VTDHC

- On-duty constraint (ODC): Each nurse cannot be assigned less than eight shifts per two weeks.

RDC is considered as a soft constraint, while the rest are hard constraints.

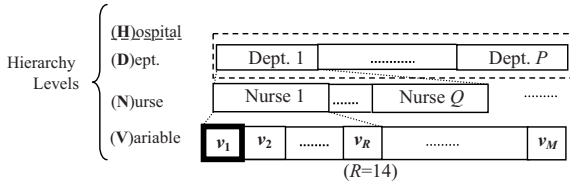
## 4.2 Constraint-Based Violation-Directed Heuristics

Violation directed operators have been already used [5, 44, 48, 49] in timetabling. Özcan [41] proposed a violation directed hierarchical hill climbing (VDHC) heuristic template to be used within MAs for solving timetabling problems and implemented an instance for solving a real-world nurse rostering problem. Experimental results show that it is a promising operator. In this study, a violation type directed hill climbing (VTDHC) heuristic template is presented as illustrated in Figure 4. VTDHC supports adaptation and co-operation of operators as a more general template than VDHC.

The VTDHC template is designed to organize a set of hill climbers where each one improves a corresponding constraint type in a given timetabling problem. A set of events among several ones is selected based on the violations. The mechanism for selecting those events is up to the user. The number of violations caused by each constraint type within the selected set is used as a guide to select a hill climber. Finally, the selected hill climber is applied onto the selected events to resolve the violations due to the related constraint type.

An *event arrangement* indicates a structured organization of events in a timetabling problem. An event arrangement will be referred to as *arrangement* in short from this point forward. It is possible to identify more than one arrangement of events for a given timetabling problem. Arrangements can be categorized as *static*, *dynamic* and *mixed*. An arrangement is labeled as static if the members in a group of variables do not change during the search process. Additionally, variables can be hierarchically organized in the static arrangements. They are logically grouped either as partitions or overlapping subsets at each hierarchy level. Static arrangement(s) can be obtained by analyzing the timetabling problem instance at hand. For example, considering NRPmh, a static arrangement of variables is derived as illustrated in Figure 5. There are four hierarchical levels within the arrangement: Hospital, Department, Nurse and Variable. Hospital is





**Fig. 5.** Static arrangement of events (shifts) for NRPmh

a group including all variables, while a group in the Nurse level is a partition, where each indicates the roster of a nurse for two weeks. In this study, the static arrangement of daily nurse shifts (events) is used as shown in Figure 5. Dynamic arrangements are based on the structure of the timetable and the assignment of events. Hence, members of a group might change during the search for an optimal solution, as the assignments of events might also change. For example, all the events (nurse shifts) scheduled at each day in a timetable constitute a dynamic arrangement of events. Mixed arrangements are a combination of both static and dynamic arrangements: e.g., events scheduled at each day in a specific department.

Combining the arrangements and VTDHC yields the design of useful hyper-heuristics. For example, VDHC represents a subset of the VTDHC heuristics, using a static arrangement of events. It is an iterative heuristic that applies a selected hill climber to a selected group of daily shifts. The hill climber selection is constraint violation-driven and based on a predetermined arrangement. First, hierarchy levels of an arrangement to be used in VDHC are decided. The top level is the starting level to operate on. As the candidate solution improves, it stays at a level. A selected hill climbing method is applied to a selected group of nurse shifts at a level, evaluating violations due to the each constraint type. VDHC restricts the area of concern to the nurse shifts at one level down in the hierarchy in the case of a relapse and the same steps are repeated. It terminates whenever no improvement is provided in none of the levels or a maximum number of steps is exceeded.

A hill climber is selected using an implicit feedback from the evolutionary process, hence VDHC is self-adjusting. During the traversal of an arrangement downwards in the hierarchy levels, VDHC switches from the individual level adaptation to the component level adaptation [52]. In this study, two other hyper-heuristics are proposed based on the VTDHC template and used within MAs. The VTDHC template can be extended and used for solving other multi-objective and/or constraint optimization problems. Moreover, heuristics based on VTDHC can be hybridized with other hyper-heuristics. In the current implementation, a single hill climber is designed for each objective. In the case of multiple hill climbers for each objective, the VTDHC instance can act as a decision mechanism for choosing the objective to improve. Then, for the improvement of a selected objective, a traditional hyper-heuristic can be utilized to choose the hill climber to employ. This is a research direction beyond the scope of this paper.

### 4.3 MAs for Solving NRPmh

For solving the NRPmh described in Section 4.1, MAs are proposed. If there are  $T$  nurses in a hospital, then the total number of biweekly shifts to be arranged is  $l = T \times 14$ , where  $l$  is chromosome length. The search space size for finding the optimal schedule becomes immense:  $3^l$ . The traditional approaches, such as branch and bound methods might even fail to obtain a solution in a reasonable amount of time, making MAs an appropriate choice. In all MAs, an allele in a chromosome represents a daily shift assignment of a nurse. Furthermore, each chromosome in the population is structured as illustrated in Figure 5.

Seven hill climbing (HC) operators are designed to be used in MAs: ENC\_HC, RDC\_HC, SHC\_HC, SNC\_HC, SDC\_HC, SSC\_HC, and ODC\_HC. Each constraint based HC operator attempts to resolve the conflicts due to the related constraint for a given variable in an individual by random rescheduling. Details of the hill climbing operators can be found in [41]. In this study, five sets of experiments are performed. In each set, a different MA is used.

In the first set of experiments, a multimeme strategy for selecting which region to apply a selected hill climber is tested. The strategy also decides how many hill climbing steps should be used. Twelve different meme values are utilized. For all problem instances used during the experiments a single acceptance strategy is used;  $b = \{1\}$  and  $n$  changes from one problem instance to another. The values in  $n$  are fixed during the start of a run as  $\{2l/4, 3l/4, l, 2l\}$ . The values of  $t$  are  $\{\text{whole, department, nurse}\}$ . A meme acting as a scheduler determines whether a hill climber will be applied to the whole individual, or to a departmental roster or to a nurse roster. Then, a constrained type is determined to be improved for the group of shifts pointed by the meme. Using a tournament selection method with a tour size of two, the constraint causing more violations within the group of shifts is favored among two randomly selected constraint types. Afterwards, the appropriate hill climber based on the selected constraint is applied to the group of shifts for a number of steps determined by the same meme. MMA experiments using this operator are performed for three different IR values. This version of MMA is labeled as MMA12. Each meme in MMA12 encodes the parameters that a hill climber requires.

During the second set of experiments, hierarchical traversal of groups is reversed in VDHC. The new hill climbing scheduler will be referred as  $r$ VDHC. Hill climbing starts from the bottom level; nurse level. As the candidate solution improves,  $r$ VDHC stays at the nurse level. A selected hill climbing method is applied in the same way as VDHC as described in Section 4.2. The  $r$ VDHC algorithm broadens the area of concern to nurse shifts in a whole department, which is one level up in the hierarchy, in the case of deterioration. Then the same steps are repeated. The termination criteria are the same as the VDHC.

In the third set of experiments a new scheduler is used. The worst nurse roster among a randomly selected two nurse rosters goes under a hill climbing process. This new scheduler is labeled as NHC. Notice that VDHC,  $r$ VDHC and NHC

**Table 4.** Experimental data set, where the number of departments and nurses are denoted as  $ndep$  and  $nnur$ , respectively. Percentage of nurses from each rank and average number of off-duty preferences of each nurse are denoted as  $pnr$  and  $avrpr$ , respectively

Label	ndep	nnur	pnr0	pnr1	pnr2	avrpr
rnd1	3	21	0.42	0.32	0.28	1.95
rnd2	3	21	0.18	0.51	0.32	0.67
rnd3	3	21	0.28	0.42	0.32	2.19
rnd4	4	21	0.14	0.47	0.42	1.67
rnd5	4	21	0.19	0.46	0.37	2.33
rnd6	4	21	0.13	0.47	0.42	0.95

can be considered as hyper-heuristics that are instances of VTDHC. In the fourth set of experiments, a multimeme algorithm is implemented. MMA uses seven memes:

$$h = \{\text{ENC\_HC}, \text{RDC\_HC}, \text{SHC\_HC}, \text{SNC\_HC}, \text{SDC\_HC}, \text{SSC\_HC}, \text{ODC\_HC}\}.$$

All the rest of the parameters are fixed:  $b = \{1\}$ ,  $t = \{\text{whole}\}$ , and  $n = \{2l\}$ . Co-evolution determines which hill climber to apply. This version of the MMA is labeled as MMA7. Each meme in MMA7 encodes only the hill climbing method to be employed. The traditional GA is used during the last set of experiments in order to evaluate the role of hill climbers.

## 5 Nurse Rostering Experiments

### 5.1 Experimental Data and Common Settings

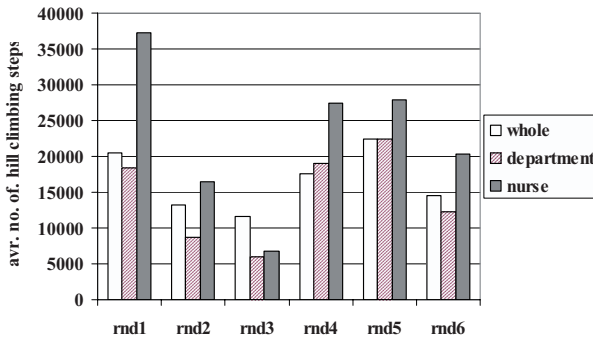
Runs are terminated whenever the overall CPU time exceeds 600 sec., or all the constraints are satisfied. The maximum number of hill climbing steps is fixed as  $2l$ . All MAs for nurse rostering use ranking as a mate selection method, giving four times higher chance to the best individual to be selected than the worst one, one point crossover and a trans-generational memetic algorithm with a replacement strategy that keeps only two best individuals from the previous generation. The mutation operator is based on the traditional approach. A shift of a nurse is randomly perturbed with a mutation probability of  $1/l$ . Based on the analysis of the NRPmh, six random problem instances are generated; rnd1-rnd6 and they are used during the experiments [41]. The characteristics of the problem instances are summarized in Table 4. The data set is publicly available at <http://cse.yeditepe.edu.tr/~eoazcan/research/TTML>.

### 5.2 Empirical Results for the NRP Experiments

The detailed experimental results of the MA with VDHC are presented in [41]. The results obtained from the first set of experiments indicate the viability of

**Table 5.** MMA12 experiments using  $IR=\{0.15, 0.20, 0.25\}$  with the random data set, where the first row denotes the success rate, the second row denotes the average number of generations per run for each IR value

IR	rnd1	rnd2	rnd3	rnd4	rnd5	rnd6
0.15	0.90	0.98	1.00	0.96	0.92	1.00
	1,145.96	217.74	77.54	697.28	667.58	234.08
0.20	0.94	0.98	1.00	0.94	0.94	1.00
	889.70	316.10	83.78	651.76	722.48	271.52
0.25	0.96	0.98	1.00	0.96	0.98	0.96
	921.12	317.62	92.20	750.18	371.34	422.90



**Fig. 6.** Average number of hill climbing steps that are executed to improve the whole set of daily shifts, a departmental roster and a nurse roster for each problem instance during the first set of experiments with MMA12, where  $IR = 0.20$ .

the MMA if used as a self-adaptive method for selecting the region where to apply a hill climber. Yet, the MA with the VDHC performs better. Experiments are repeated for different values of IR around 0.20. The results are summarized in Table 5 for the experimental data. No IR value is significant. Considering the average success rates, all IR values yield almost the same performance. An interesting result of the first set of experiments is that MMA12 selects mostly a nurse roster and then applies a hill climber to it, as illustrated in Figure 6 for  $IR = 0.20$ . The rest of the experiments are performed on Pentium IV 3 GHz machines with 2 GB RAM.

During the following experiments, MAs with  $r$ VDHC, NHC, simple GA and MMA7 are tested. The success rate of each algorithm for each problem instance is presented in Table 6. Obviously, hill climbing boosts the performance GAs. The simple genetic algorithm turns out to be the worst algorithm for solving the problem instances. In almost none of the runs is a violation-free schedule obtained. The empirical results yield the success of MAs with the following hyper-heuristics from the best towards the worst: VDHC,  $r$ VDHC and NHC.

**Table 6.** The success rates of different algorithms for solving random problem instances

Label	VDHC	rVDHC	NHC	MMA7 (IR = 0.20)	MMA12 (IR = 0.20)	Simple GA
rnd1	0.96	0.94	0.68	0.86	0.94	0.00
rnd2	1.00	0.98	0.88	0.96	0.98	0.04
rnd3	1.00	1.00	0.98	1.00	1.00	0.00
rnd4	0.98	0.94	0.28	0.18	0.94	0.00
rnd5	1.00	0.86	0.26	0.30	0.94	0.00
rnd6	1.00	1.00	0.68	0.50	1.00	0.00
Avr.	0.99	0.95	0.63	0.63	0.97	0.01

The performances of MMA7 and MMA12 are comparable to the performances of NHC and VDHC, respectively. Results show that letting the multimeme algorithm choose the region where to apply a constraint-based hill climber based on a static hierarchical arrangement of events (MMA12) performs better than letting it choose the meme for solving nurse rostering problem instances (MMA7).

## 6 Conclusions

Memetic algorithms, including the self-generating multimeme memetic algorithms proposed by Krasnogor [33] are investigated. Different MAs are experimented using a set of benchmark functions and nurse rostering problem instances, generated randomly by Özcan [41] based on a real-world nurse rostering problem. Some common empirical results are obtained from both investigations. As expected, the performance of a genetic algorithm improves if a hill climbing operator is also utilized. The Lamarckian learning mechanism employed by MMAs yields appealing results for selecting a meme among the set of memes during the evolutionary process. Yet, MAs with a good meme choice perform better. Different memes yield different performances. In the benchmark experiments, MMAs identify the useful memes for all functions, but unfortunately, no synergy between the hill climbers is noticed during the search.

The following observations are gathered from the benchmark function experiments. The steepest descent hill climbing performs well in noisy and deceptive search landscapes, while both Davis's bit hill climber and the next descent hill climber are successful in locating optima in multimodal search landscapes. If the search landscape contains plateaus, then it seems that Davis's bit hill climber performs slightly better. Furthermore, the average performance of the Davis's bit hill climbing is the best over all benchmark functions.

MAs are very promising approaches for tackling nurse rostering problems. The proposed heuristic template combined with a prior knowledge about a timetabling problem, such as a static arrangement, provides a promising guide for designing adaptive heuristics. Hyper-heuristics enable the utilization of multiple hill climbers under a single black box mechanism. Such a hyper-heuristic

that always accepts an improving move can be embedded into an MA acting as if a single hill climber without changing the framework of both approaches. MAs, each containing such an instance are compared to the MMAs, each using different hill climber settings. The empirical results indicate the success of the MA with VDHC [41] over the rest of the MAs presented in this paper. MMA12 delivers a matching performance. VDHC using tournament selection provides a better co-operation among constraint-based memes. The hierarchical traversal over the groups based on a static arrangement during the hill climbing seems to work as well. Applying a constraint-based meme to a larger group of events first and then narrowing the area of concern generates better results than the reverse traversal. Still, *r*VDHC shows potential. The experimental results on the nurse rostering problems show that the choice of operator parameters that are encoded into a meme affects the performance of MMA. Similarly, the choice of hyper-heuristic for managing a set of hill climbers to be used within MA has an influence on its performance.

*Acknowledgement.* This research is supported by TUBITAK (The Scientific and Technological Research Council of Turkey) under the grant number 105E027.

## References

1. Ackley, D.: An empirical study of bit vector function optimization. In: Davis, L. (ed.) Genetic Algorithms and Simulated Annealing, pp. 170–215. Pitman, London (1987)
2. Ahmad, J., Yamamoto, M., Ohuchi, A.: Evolutionary algorithms for nurse scheduling problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 196–203 (2000)
3. Aickelin, U., Bull, L.: On the application of hierarchical coevolutionary genetic algorithms: recombination and evaluation partners. *Journal of Applied Systems Studies* 4, 2–17 (2003)
4. Aickelin, U., Dowsland, K.: An indirect genetic algorithm for a nurse scheduling problem. *Computers and Operations Research* 31, 761–778 (2003)
5. Alkan, A., Özcan, E.: Memetic algorithms for timetabling. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1796–1802 (2003)
6. Berrada, I., Ferland, J., Michelon, P.: A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Science* 30, 183–193 (1996)
7. Burke, E.K., Cowling, P.I., De Causmaecker, P., Vanden Berghe, G.: A memetic approach to the nurse rostering problem. *Applied Intelligence* 15, 199–214 (2001)
8. Burke, E.K., De Causmaecker, P., Petrovic, S., Vanden Berghe, G.: Variable neighbourhood search for nurse rostering problems. In: Resende, M.G.C., de Sousa, J.P. (eds.) *Metaheuristics: Computer Decision-Making*, ch. 7, pp. 153–172. Kluwer, Dordrecht (2003)
9. Burke, E.K., De Causmaecker, P., Vanden Berghe, G.: A hybrid tabu search algorithm for the nurse rostering problem. In: McKay, B., Yao, X., Newton, C.S., Kim, J.-H., Furuhashi, T. (eds.) *SEAL 1998. LNCS (LNAI)*, vol. 1585, pp. 187–194. Springer, Heidelberg (1999)

10. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The state of the art of nurse rostering. *Journal of Scheduling* 7, 441–499 (2004)
11. Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyperheuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 457–474. Kluwer, Dordrecht (2003)
12. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9, 451–470 (2003)
13. Chun, A.H.W., Chan, S.H.C., Lam, G.P.S., Tsang, F.M.F., Wong, J., Yeung, D.W.M.: Nurse rostering at the Hospital Authority of Hong Kong. In: *Proceedings of the 17th National Conference on AAAI and 12th Conference on IAAI*, pp. 951–956 (2000)
14. Cowling, P., Kendall, G., Soubeiga, E.: A hyper-heuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
15. Davis, L.: *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York (1991)
16. Davis, L.: Bit climbing, representational bias, and test suite design. In: *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 18–23 (1991)
17. De Jong, K.: An analysis of the behaviour of a class of genetic adaptive systems. Ph.D. Thesis, University of Michigan, Ann Arbor, MI (1975)
18. Dowsland, K.: Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operations Research* 106, 393–407 (1998)
19. Duenas, A., Mort, N., Reeves, C., Petrovic, D.: Handling preferences using genetic algorithms for the nurse scheduling problem. In: *MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, Nottingham, vol. 1, pp. 180–196 (August 2003)
20. Easom, E.E.: A survey of global optimization techniques. M.Eng. Thesis, University of Louisville, KY (1990)
21. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing* 5, 691–703 (1976)
22. Fang, H.L.: Genetic algorithms in timetabling and scheduling. Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh, Scotland (1994)
23. Gendreau, M., Buzon, I., Lapierre, S., Sadr, J., Soriano, P.: A tabu search heuristic to generate shift schedules. In: *MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, Nottingham, vol. 2, pp. 526–528 (August 2003)
24. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA (1989)
25. Goldberg, D.E.: Genetic algorithms and Walsh functions: part I, a gentle introduction. *Complex Systems* 3, 129–152 (1989)
26. Goldberg, D.E.: Genetic algorithms and Walsh functions: part II, deception and its analysis. *Complex Systems* 3, 153–171 (1989)
27. Griewangk, A.O.: Generalized descent of global optimization. *Journal of Optimization Theory and Applications* 34, 11–39 (1981)
28. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI (1975)

29. Han, L., Kendall, G.: Application of genetic algorithm based hyper-heuristic to personnel scheduling problems. In: Kendall, G., Burke, E.K., Petrovic, S., Gendreau, M. (eds.) MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications, Nottingham, August 2003, pp. 528–537. Springer, Berlin (2005)
30. Kawanaka, H., Yamamoto, K., Yoshikawa, T., Shinogi, T., Tsuruoka, S.: Genetic algorithms with the constraints for nurse scheduling problem. In: Proceedings of IEEE Congress on Evolutionary Computation, CEC, Seoul, pp. 1123–1130 (2001)
31. Krasnogor, N.: Studies on the theory and design space of memetic algorithms. Ph.D. Thesis, University of the West of England, Bristol, UK (2002)
32. Krasnogor, N., Smith, J.E.: Multimeme algorithms for the structure prediction and structure comparison of proteins. In: GECCO 2002. Proceedings of the Bird of a Feather Workshops, pp. 42–44 (2002)
33. Krasnogor, N., Smith, J.E.: Emergence of profitable search strategies based on a simple inheritance mechanism. In: GECCO 2001. Proceedings of the Genetic and Evolutionary Computation Conference, pp. 432–439 (2001)
34. Krasnogor, N., Smith, J.E.: A memetic algorithm with self-adaptive local search: TSP as a case study. In: GECCO 2000. Proceedings of the Genetic and Evolutionary Computation Conference, pp. 987–994 (2000)
35. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards* 84, 489 (1979)
36. Li, H., Lim, A., Rodrigues, B.: A hybrid AI approach for nurse rostering problem. In: Proceedings of the 2003 ACM Symposium on Applied Computing, pp. 730–735 (2003)
37. Mitchell, M., Forrest, S.: Fitness landscapes: royal road functions. In: Baeck, T., Fogel, D., Michalewicz, Z. (eds.) *Handbook of Evolutionary Computation*, Institute of Physics Publishing, Bristol, and Oxford University Press, Oxford (1997)
38. Moscato, P., Norman, M.G.: A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: Valero, M., Onate, E., Jane, M., Larriba, J.L., Suarez, B. (eds.) *Parallel Computing and Transputer Applications*, pp. 177–186. IOS Press, Amsterdam (1992)
39. Ning, Z., Ong, Y.S., Wong, K.W., Lim, M.H.: Choice of memes in memetic algorithm. In: Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems (2003)
40. Ong, Y.S., Keane, A.J.: Meta-Lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation* 8, 99–110 (2004)
41. Özcan, E.: Memetic Algorithms for Nurse Rostering. In: Yolum, p., Güngör, T., Gürgen, F., Özturan, C. (eds.) *ISCIS 2005. LNCS*, vol. 3733, pp. 482–492. Springer, Heidelberg (2005)
42. Özcan, E.: Towards an XML based standard for timetabling problems: TTML. In: Kendall, G., Burke, E.K., Petrovic, S., Gendreau, M. (eds.) MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications, Nottingham, p. 163. Springer, Berlin (August 2005)
43. Özcan, E., Bilgin, B., Korkmaz, E.E.: Hill climbers and mutational heuristics in hyperheuristics. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) *Parallel Problem Solving from Nature - PPSN IX. LNCS*, vol. 4193, pp. 202–211. Springer, Heidelberg (2006)
44. Özcan, E., Ersoy, E.: Final exam scheduler – FES. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1356–1363 (2005)



45. Özcan, E., Onbasioglu, E.: Memetic algorithms for parallel code optimization. *International Journal of Parallel Programming* 35, 33–61 (2007)
46. Radcliffe, N.J., Surry, P.D.: Formal memetic algorithms. In: Fogarty, T.C. (ed.) *Evolutionary Computing*. LNCS, vol. 865, pp. 1–16. Springer, Heidelberg (1994)
47. Rastrigin, L.A.: *Extremal Control Systems, Theoretical Foundations of Engineering Cybernetics Series*, Nauka, Moscow (1974)
48. Ross, P., Corne, D., Fang, H.-L.: Improving evolutionary timetabling with delta evaluation and directed mutation. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) *Parallel Problem Solving from Nature - PPSN III*. LNCS, vol. 866, pp. 556–565. Springer, Heidelberg (1994)
49. Ross, P., Corne, D., Fang, H.-L.: Fast practical evolutionary timetabling. In: *Proceedings of the AISB Workshop on Evolutionary Computation*, pp. 250–263 (1994)
50. Schwefel, H.-P.: *Numerical Optimization of Computer Models*. Wiley, Chichester (1981)
51. Schwefel, H.-P.: *Evolution and Optimum Seeking*. Wiley, New York (1995)
52. Smith, J., Fogarty, T.C.: Operator and parameter adaptation in genetic algorithms. *Soft Computing* 1, 81–87 (1997)
53. Tasoulis, D., Pavlidis, N., Plagianakos, V., Vrahatis, M.: Parallel differential evolution. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 2023–2029. IEEE Computer Society Press, Los Alamitos (2004)
54. Whitley, D.: Fundamental principles of deception in genetic search. In: Rawlins, G.J.E. (ed.) *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA (1991)

# An Evaluation of Certain Heuristic Optimization Algorithms in Scheduling Medical Doctors and Medical Students

Christine A. White<sup>1</sup>, Emilina Nano<sup>2</sup>, Diem-Hang Nguyen-Ngoc<sup>2</sup>,  
and George M. White<sup>2</sup>

<sup>1</sup> Department of Medicine, Division of Nephrology,  
Queen's University, Kingston, Canada K7L 2V6  
cw38@post.queensu.ca

<sup>2</sup> School of Information Technology and Engineering,  
University of Ottawa, ON, Canada K1N 6N5  
white@site.uottawa.ca

**Abstract.** Four heuristic algorithms based on or inspired by the well-known Tabu Search method have been used to cast heuristically optimized schedules for a clinical training unit of a hospital. It has been found experimentally that the algorithm of choice for this problem depends on the exact goal being sought where the execution time is one of the components of the goal. If only one run is allowed, then classical Tabu Search with a tenure of 5 gave the schedule with the lowest average (and fixed) penalty. If time is not of concern and many runs are allowed then the Great Deluge algorithm may generate the schedule with the lowest penalty.

## 1 Introduction

The scheduling of personnel can often be accomplished in two phases, the phase that deals with time-of-day or shift scheduling, and the phase that deals with day-of-week scheduling. Baker [2] has named these types of labour scheduling *tour scheduling*.

Glover and McMillan [7] have reviewed the problems of employee scheduling while more specialized reviews of the tour scheduling literature have been published by Alfares [1] and by Ernst et al. [5]. It is one thing to find *feasible* schedules, i.e. schedules that satisfy all the staffing rules, but quite another to find an *optimal* feasible schedule, i.e. one that not only satisfies the rules but also minimizes (or maximizes) some objective function.

In work described in a previous PATAT conference [9] a stand-alone system for casting schedules of medical staff in the Internal Medicine Clinical Teaching Unit of the Ottawa Hospital was built using the Java programming language. The algorithm constructed an initial feasible schedule and then heuristically optimized it to reduce its perceived 'badness'. The algorithm used was a simple version of the tabu search (TS) algorithm introduced by Glover [6] and used many times since.

The requirement was to produce duty rosters (locally referred to as *call schedules*) for medical trainees (residents and medical students) in the Clinical Teaching Unit to man the overnight shift. The duties of a shift consist in rendering medical assistance to patients in need of it during the night when the majority of the medical trainees are no longer on duty. For each night in a 28-night cycle, a shift of (ideally) 5 persons consisting of a senior resident, 2 junior residents and 2 medical students has to be scheduled. Because of chronic understaffing the shift often consists of fewer than 5 persons. Sometimes 4 and sometimes 3 persons are used if there is not enough staff available. The staff chosen for these shifts have various ‘ranks’ and may belong to one of two teams. Since these evening rounds are in addition to regular day shifts that the medical trainees must work, there are very stringent requirements that prohibit the personnel from being overworked beyond a certain point. These numerous requirements are formulated as *soft constraints* and each violation of a constraint is associated with an integer penalty whose magnitude is a measure of the undesirability of relaxing that constraint. The sum of these integers is the measure of the ‘badness’ of the schedule. The TS algorithm is used to minimize this badness. An example of a call schedule is shown in Figure 1.

This schedule consists of a duty roster of exactly 28 days in length, each day showing the assignment of 5 or fewer medical staff members. Ideally there will be exactly 5 members but for financial reasons, only 4 or 3 may actually be scheduled. Each schedule begins on a Tuesday and ends on a Monday. The first line of Figure 1 shows that the senior resident is Dr Zaidi. He will be in charge of the unit on the first Tuesday night. He is assisted by three other persons, Shefrin, Carrier and Puglia. A fifth person is not available on that night. Some of these assistants are junior doctors while others are medical students.

The senior resident is the doctor in charge. The two teams, A and B, consist of a ‘First Call’, i.e. the first person to call if required, and a ‘Second Call’, the next person to call (if there is one). The members of a team work closely together.

Any call schedule has an associated penalty that quantifies how ‘bad’ it is. The components of this penalty can be broadly classified as horizontal or vertical penalties. Some nights, e.g. the third and fourth lines on the schedule, corresponding to Thursday and Friday of the first week, are fully staffed with appropriate members from each team. Other nights, e.g. the final Thursday are very short staffed, with only three medical personnel on duty. The first of these examples attracts a penalty of 0 while the second example attracts a penalty of 300. These are examples of horizontal penalties. They can be evaluated simply by scanning each line separately. Table 1 lists the various defects that each night’s shift might have and the corresponding penalties.

Vertical penalties are those that arise from consecutive nights. The first Thursday and Friday of Figure 1 has one trainee, Oliveira, working for two consecutive nights. This attracts a penalty of 100. The weekends, defined to consist of Friday, Saturday and Sunday, are very sensitive. A pattern consisting of working on Friday and Sunday (but *not* Saturday) or its converse are greatly desired.

	Team A			Team B	
	Senior	1st Call	2nd Call	1st Call	2nd Call
Tue:	Zaidi	Shefrin	Carrier	Puglia	-----
Wed:	ElFirjani	Mongiardi	-----	Rajput	Mufti
Thu:	Jolicoeur	Marwaha	Payne	Oliveira	Cohn
Fri:	Ellen	Mongiardi	Carrier	Oliveira	Bal
Sat:	Zaidi	Taylor	Radke	Rajput	-----
Sun:	Ellen	Mongiardi	Carrier	Oliveira	Bal
Mon:	ElFirjani	Taylor	-----	Puglia	Mufti
Tue:	Treki	Shefrin	Payne	Puglia	-----
Wed:	Stewart	Taylor	Carrier	Bal	-----
Thu:	ElFirjani	Mongiardi	Radke	Rajput	Cohn
Fri:	Jolicoeur	Taylor	-----	Puglia	Mufti
Sat:	Stewart	Mongiardi	Payne	Oliveira	Bal
Sun:	Jolicoeur	Taylor	-----	Puglia	Mufti
Mon:	Treki	Marwaha	Radke	Rajput	Cohn
Tue:	Stewart	Taylor	Payne	Bal	-----
Wed:	Jolicoeur	Shefrin	Carrier	Oliveira	-----
Thu:	Ellen	Marwaha	-----	Oliveira	Mufti
Fri:	Zaidi	Shefrin	Radke	Rajput	-----
Sat:	Jolicoeur	Marwaha	Carrier	Cohn	-----
Sun:	Zaidi	Shefrin	Radke	Rajput	-----
Mon:	Ellen	Mongiardi	-----	Rajput	Mufti
Tue:	ElFirjani	Marwaha	Payne	Puglia	Cohn
Wed:	Zaidi	Shefrin	Radke	Oliveira	-----
Thu:	Treki	Taylor	-----	Bal	-----
Fri:	Stewart	Marwaha	Payne	Cohn	-----
Sat:	Ellen	Shefrin	-----	Puglia	Mufti
Sun:	Stewart	Marwaha	Payne	Cohn	-----
Mon:	ElFirjani	Mongiardi	Radke	Bal	-----

**Fig. 1.** An example of a call schedule

Failure to achieve this attracts a high penalty. The example of Table 1 manages to achieve this goal at the expense of attracting horizontal penalties. As a rule, horizontal and vertical penalties play against each other, reducing one usually implies increasing the other. The one that ‘wins’ is the one having the lower value.

In the quest to cast the best schedule in a reasonable time, four different heuristic algorithms based on the local search strategy were implemented and tested with real data obtained from the hospital. A number of different data sets were used. This paper summarizes the four algorithms used and discusses the results obtained from each one using a single representative data set.

**Table 1.** Conditions and their penalties

Condition	Penalty
One student missing	5
Student replaces missing junior - senior is on student's team	10
Two students missing	20
Junior and student missing - student takes junior's place – senior is on student's team	40
Student replaces missing junior – senior is not on student's team	80
Junior and student missing – student takes junior's place – senior is not on student's team	100
Two juniors missing - replaced by two students	300
Any other defect	500

## 2 Algorithms Investigated

The algorithms investigated were:

- Tabu Search with Fixed Tenure
- Tabu Search with Random Tenure
- Great Deluge
- IDWalk.

The first of these is deterministic. The algorithm always yields the same answer when given the same data. Thus the Tabu Search with fixed tenure was executed once for each tenure value. The other algorithms are not deterministic. A pseudo-random generator is used and therefore each run may yield a different result. For these cases, 20 runs were made and the values shown in the tables are based on these 20 runs. For the Great Deluge algorithm, an additional 1800 runs were made and analysed more thoroughly.

The results were all obtained from code written in C# on the Microsoft Visual Studio.Net IDE and executed on a PC under Windows XP with a Celeron chip at 2.8 GHz. with 192 MBytes of RAM.

### 2.1 Tabu Search with Fixed Tenure

The classical Tabu Search algorithm whose entries in the single tabu list have a fixed tenure was the original algorithm implemented in the project [9].

An initial solution is generated by a combination of constraint logic that satisfies the weekend requirement followed by a simple bin packing procedure that satisfies the requirements of the rest of the week and considers vacation and rank factors. The weekend requirement specifies that a Friday and the following Sunday plus a Saturday from a different weekend be assigned to staff where possible. This would be very simple to do if it were not for holidays, vacations

and days off which can occur any time. The weekend requirement is enforced by a simple chronological backtracking algorithm. The weekday slots are then filled in turn by finding a suitable staff member, not already working, who has not yet completed the required number of calls.

Then a multiphase tabu search is performed that heuristically reduces a penalty function by considering the seniors, juniors and students in sequence. A *neighbour* of a given solution is another solution that can be obtained by a small perturbation of that solution. A *neighbourhood* is the set of such neighbours. The basic move that generates a new neighbourhood is a 'swap', exchanging the places of two seniors, juniors or students as appropriate. As there are 5 persons on duty during a call, there are 5 sequences generated, one for the seniors, one for the juniors on team A, one for the juniors on team B, one for the students on team A and one for the students on team B. The entire space was partitioned into 5 sub-spaces to reduce the amount of time required to complete an evaluation of the neighbourhood. Since there are 28 days in a schedule, there are  $28 \times 27/2 = 378$  possible swaps, each of which must be evaluated for each tabu move. This is a quantity that is easily handled by the system used.

A tabu minimization is performed using each of the five neighbourhood sub-spaces in turn. When this is completed, the schedule is examined to discover whether the solution could be improved by *removing* one of the students. In this environment, more is not always better. In spite of the chronic understaffing of the hospital, if there are too many students on duty and too few seniors and juniors to supervise their work, the situation is deemed worse than if there were fewer students. Therefore, during this part of the algorithm, surplus students are removed from the schedule if this would reduce the total schedule penalty.

At this point, the neighbourhood is redefined. The juniors and students from team A and the juniors and students of team B are joined to produce a larger neighbourhood created by redefining the *move* to be a *rotation* across two rows of the schedule. This involves changing the 'slots' of four persons. This is done for the A team followed by the B team.

Finally, the original moves and neighbourhoods are restored and the tabu optimization is recycled through the five neighbourhoods until no further improvement can be found. In each case, the search is terminated after 150 failed attempts to improve the best solution found so far. This value was chosen by experimenting with different values. Values lower than this were found to give worse results. Values higher than this tended to give the same or similar results but took longer to get there.

This algorithm was run four times with the real hospital data keeping the tenure fixed during a run but varying its value in different runs. This algorithm is deterministic. A run always returns the same schedule having the same penalty when given the same data. The aspiration function used by TS was set to return a value equal to the best value found so far. The algorithm was run using fixed tenures of 5, 10, 20 and 40. The values of the penalties and execution times of the schedules obtained are shown in Table 2.

**Table 2.** Values of penalties and execution times

Tenure (fixed)	Penalty	Execution time (s)
40	1520	27.6
20	1495	27.0
10	1415	25.8
5	1270	25.9

**Table 3.** Statistics obtained from runs of TS with random tenure

	Penalty	Execution time (s)
Min	1235	26.4
Max	1495	28.5
Mean	1342	27.2
SD	82.8	0.6

## 2.2 Tabu Search with Random Tenure

This variation of the classical TS differs only in that the tenure of an entry in the tabu list is drawn from a series of pseudo-random integers whose value is determined when the entry is inserted into the list. This procedure is the same as that discussed by Di Gaspero and Schaerf in [3]. The distribution of these tenures was uniform discrete on the interval [10,40]. The interval [5,40] was initially selected since the fixed tenure algorithm yielded good results for a tenure of length 5. However, with this interval, the results of the runs were very nearly all the same, most of the values being the same as that found for the fixed tenure case of length 5. Evidently for the data used, the random tenure algorithm tended to degenerate to the fixed tenure case. For all further studies, the interval [10,40] was used.

After examining a neighbourhood, the best of the solutions is taken as the solution to start with in the next round and the previous solution is placed in the tabu list with tenure calculated at the point of insertion. This algorithm is not deterministic and successive runs with the same program and the same data usually give different schedules having different penalties. Statistics obtained from the 20 runs are summarized in Table 3.

## 2.3 Great Deluge

This method was introduced by Gunter Dueck in 1993 [4]. The name ‘Great Deluge’ was chosen by Dueck to illustrate the progress of the algorithm in an analogy where a person is trying to keep his feet dry by climbing in mountainous terrain during a great deluge. The person moves by taking a step in some

**Table 4.** Statistics obtained from runs of the Great Deluge algorithm

	Penalty	Execution time (s)
Min	1210	7.4
Max	2840	9.6
Mean	1720	7.8
SD	353.9	0.6

randomly chosen direction while the water level continues to rise. He stays in the new position only if he can keep his feet dry. If this is not possible he moves randomly again. Eventually all moves result in wet feet or the time has run out and the algorithm stops. This algorithm has the desirable property that there is only one adjustable parameter, the rate of rise of the water level. In the implementation the water level is initialized to the value of the initial solution, and *decreases* by a value of 2 with each iteration. The algorithm terminates when no further moves are possible. As above, this method is not deterministic. The results are summarized in Table 4.

## 2.4 IDWalk

This method, called Intensification/Diversification Walk (or IDWalk), was introduced by Neveu et al. [8] and is related to the TS method. There are three parameters,  $S$ , the number of moves,  $Max$ , the number of potential neighbours studied in each move and  $SpareNeighbour$ , the diversification strategy.

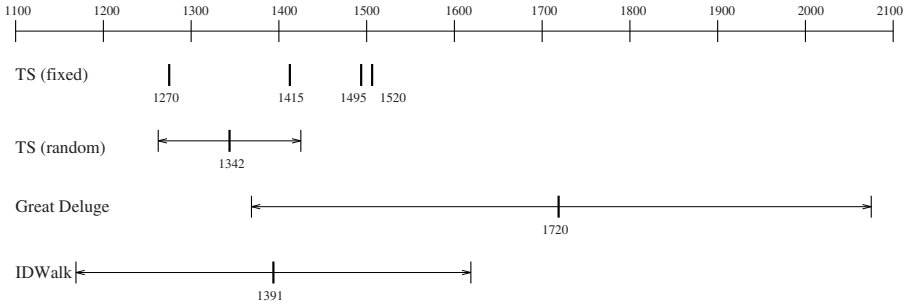
This algorithm performs  $S$  moves and returns the best solution it found. In choosing the next move to make, it examines at most  $Max$  candidate neighbours, selecting them randomly. If the penalty of a candidate,  $x'$ , is less than or equal to the penalty of the current solution, then the solution corresponding to  $x'$  is chosen for the move. If no neighbour has been selected from among the  $Max$  examined, then one of the rejected candidates is chosen for the next move. If  $SpareNeighbour$  was set equal to 'best', then the least bad of the rejected neighbours is chosen for the next move. If  $SpareNeighbour$  was set to 'any' then any one of the rejected neighbours is chosen randomly for the next move.

In this investigation,  $S = 1000$  and  $Max = 378$ . The two possible choices for  $SpareNeighbour$  were tried and it was found that the value, 'best' gave the superior results. No systematic study of various values of these parameters was made at this time. Investigation of these factors is ongoing. The values selected were tested and found to give good results but is as yet unknown whether a different choice of the parameters  $S$  and  $Max$  would yield superior results. As before a number of runs were made using the same input data. The results are summarized in Table 5.



**Table 5.** Statistics obtained from runs of the IDWalk algorithm

	Penalty	Execution time (s)
Min	1160	57.4
Max	1810	77.5
Mean	1391	68.1
SD	225.1	7.7



**Fig. 2.** Comparison of results

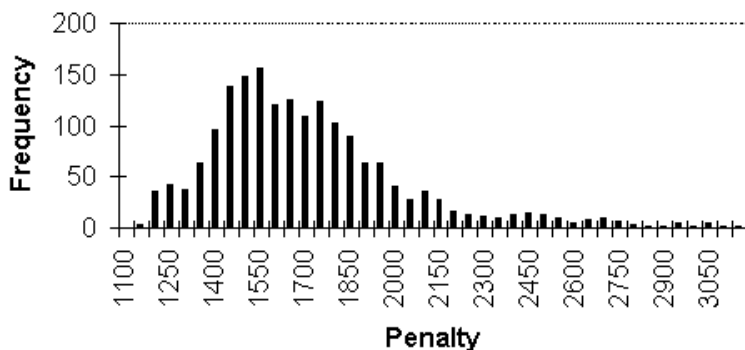
### 3 Comparison of Penalties Obtained by the Four Methods

These results are shown graphically in Figure 2.

The scale of the entire figure is shown by the top line. The left end of the line has the value 1100 and the right end has the value 2100. The second line of the figure shows the penalties when TS is used with fixed tenures of 5, 10, 20 and 40. The best call schedule (with the smallest penalty) is found when the tenure was fixed at 5. This schedule has a penalty of 1270.

When random tenures in the interval  $[10,40]$  are used, the schedules obtained are generally better than those obtained with a fixed tenure set to any value within this interval. Recall that about 66% of the values obtained are within one standard deviation on either side of the mean (which means that about 33% are not). The best value obtained was 1235 which is better than anything that was obtained using fixed tenure. The worst value was 1495 which is about the same value as that obtained using a fixed tenure of 20 and better than the results with a value of 40. The mean value and the interval delineated by plus or minus one standard deviation (containing about 66% of the values) is shown in Figure 2 immediately below the results for TS (fixed).

As expected, the values obtained with the Great Deluge algorithm were rather poor. The average penalty value of 1720 was the worst of the four methods



**Fig. 3.** Histogram of 1800 penalties obtained for the Great Deluge algorithm

and the standard deviation was the largest obtained. The spectrum of values obtained was such that the worst was worse than any of the TS results but the best was also better than anything found by TS. The algorithm executed rapidly and its mean time to completion, at 7.8 seconds, was better than three times faster than its fastest rival. The mean value and the one standard deviation interval are shown in Figure 2.

The IDWalk method yielded a mean penalty of 1391, better than Great Deluge but worse than TS with random tenure. The large standard deviation of about 225 illustrates the range of values obtained. Its lowest value of 1160 was the best value obtained by any of the algorithms. Its highest value was higher than any obtained by any of the TS methods but better than Great Deluge. Its mean execution time of 68.1 s makes it the slowest algorithm to complete execution. The lowest value was obtained by the run having the longest execution time. As before, the mean value and the one standard deviation interval are shown in the last line of the figure.

A *t*-test on pairwise comparison of the three distributions assuming unequal variances shows that the TS (random) and IDWalk algorithms produce results such that the difference between their mean values is not significant at the 95% level. When TS (random) and Great Deluge are compared the means were found to be significantly different. The same was true when the IDWalk and Great Deluge distributions were compared.

Because of the wide range of penalty values found by the Great Deluge algorithm, this case was studied further. Using the same input data, 1800 runs were made. The histogram of the results obtained is shown in Figure 3.

It is evident that there is a large variation in the values obtained, the smallest value being 1135 (obtained 3 times) and the largest value being 3620 (obtained one). Therefore the worst schedule has an associated penalty that is more than three times larger than the best one. The penalty of 1135 is the best obtained in any run of any algorithm studied here. Although the mean value was the worst obtained in this study, the minimum value was the best minimum value due to the high variability of the results.

## 4 Discussion

Using the experimental results obtained using data from this medical call schedule problem, the ranking of the four methods tested based on their mean penalty is:

1. TS – fixed tenure = 5
2. TS with random tenure [10,40]
3. IDWalk
4. Great Deluge.

For consistency of results, the method of choice is TS with random tenure for the non-deterministic algorithms and TS with fixed tenure of 5 for the deterministic ones.

If a good solution must be obtained within about a minute's time, then TS with fixed tenure = 5 is the best bet. However, if execution time is not critical, the method of choice may be the Great Deluge which yields individual results rapidly, even if most of them are not very good. One of them may be very good indeed, although a long series of runs may be required to find it.

The same may be true of IDWalk although this was not investigated here. For the time of one run of IDWalk, we could have about ten runs of Great Deluge. This may be the method of choice in a setting where actual schedules have to be produced and used in a real hospital. The system can be left running overnight and the best of the schedules obtained can be retrieved in the morning. With 12 hours available for repeated automatic runs and with Great Deluge's 8 second run time, about 5400 schedules can be produced and the best one used. If desired, the best schedule could be taken as the initial solution for another metaheuristic to further improve.

It should be remembered that these results strongly reflect the specific requirements and data taken from one institution. Discussions with other units in the same hospital revealed that apart from having to use different data the algorithms would have to use different weights and different criteria in forming the corresponding penalties. This might lead to different conclusions. We are investigating this further.

## 5 Conclusions

Depending on the desired goals and available execution time, the algorithm of choice for this problem will vary. The lowest average penalty is obtained by TS with fixed tenure of 5. The lowest individual penalty of all was obtained during the additional Great Deluge runs. If time is of little concern, multiple runs of Great Deluge may be the best strategy. It is ironic that the algorithm that produces the distribution with the *worst* mean is the same one that produces the *best* individual value.

## References

1. Alfares, K.: Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research* 127, 145–175 (2004)
2. Baker, K.: Workforce allocation in cyclic scheduling problems. *Operations Research Quarterly* 27, 155–167 (1976)
3. Di Gaspero, L., Schaerf, A.: Multi-neighbourhood local search with application to course timetabling. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 262–275. Springer, Heidelberg (2003)
4. Dueck, G.: New optimization algorithms. *Journal of Computational Physics* 104, 86–92 (1993)
5. Ernst, A., Jiang, H., Krishnamoorthy, M., Dier, D.: Staff scheduling and rostering: a review of applications, methods and modelling. *European Journal of Operational Research* 153, 3–27 (2004)
6. Glover, F., Laguna, M.: *Tabu Search*. Kluwer, Dordrecht (1997)
7. Glover, F., McMillan, C.: The general employee scheduling problem: an integration of management science and artificial intelligence. *Computers and Operations Research* 13, 563–593 (1986)
8. Neveu, B., Trombettoni, G., Glover, F.: Idwalk: A Candidate List Strategy With a Simple Diversification Device. In: Wallace, M. (ed.) *CP 2004*. LNCS, vol. 3258, pp. 423–437. Springer, Heidelberg (2004)
9. White, C.A., White, G.M.: Scheduling Doctors For Clinical Training Unit Rounds Using Tabu Optimization. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 120–128. Springer, Heidelberg (2003)

# **Timetabling of Meetings**

# Scheduling Research Grant Proposal Evaluation Meetings and the Range Colouring Problem

Patrick Healy

CS Department, University of Limerick, Limerick, Ireland  
patrick.healy@ul.ie

**Abstract.** In many funding agencies a model is adopted whereby a fixed panel of evaluators evaluate the set of applications. This is then followed by a general meeting where each proposal is discussed by those evaluators assigned to it with a view to agreeing on a consensus score for that proposal. It is not uncommon for some evaluators to be unavailable for the entire duration of the meeting; constraints of this nature, and others, complicate the search for a solution and take it outside the realm of the classical graph colouring problem. In this paper we (a) report on a system developed to ensure the smooth running of such meetings and (b) compare two different ILP formulations of a sub-problem at its core, the list-colouring problem.

## 1 Background

The process of evaluating research grant proposals presents some interesting opportunities for operations research practitioners and researchers. The model we discuss in this paper assumes that a fixed pool of evaluators exists and a set of grant proposals is distributed amongst them subject to the evaluators' stated abilities to evaluate each. Several constraints affect the allocation of proposals to evaluators. For example, insofar as it is reasonable, all evaluators should be allocated the same load, although some evaluators, such as vice-chairs, may be expected to take a reduced load of evaluations due to other duties; upper limits may be enforced on the number of evaluators employed; in the interests of (particularly, novice) evaluators not being intimidated at the associated meetings that occur later, no proposal should have a majority of vice-chairs evaluating it; for each proposal, one evaluator should be appointed as proposal reporter amongst the – usually 3, although requests for larger financial sums necessitate more – evaluators assigned to it with this extra duty evenly allocated amongst all evaluators.

These complications, and other side constraints, make the allocation problem an interesting study in its own right. Nonetheless, for present purposes, we will assume that evaluations have been completed and the evaluators' recommendations must now be unified in to an ordered *ranking list*. From this list grant applications will be offered funding subject to budget and other administrative considerations. Although other procedures for arriving at a ranking list are possible [5] the most commonly used procedure appears to be that each

evaluator initially assigns a numerical mark to each proposal allocated to them. At a follow-up meeting of the (usually three) evaluators associated with each proposal a consensus mark is arrived at. The ranking list is then based on the consensus marks for each proposal.

Our interest begins when a general panel meeting (a panel may be thought of as a general research area, e.g. computer science) brings all of the evaluators together. Each proposal is discussed face to face by the assigned evaluators for the purpose of agreeing a consensus position and category scores, from which the final evaluation report may be written; the consensus meeting runs for a fixed length of time. Following the entry of all scores in a database, a ranking list is generated which forms the basis for funding decisions by the grant agency. So that the entire panel of evaluators may agree to the ranking list it is desirable that all consensus meetings be completed as quickly as possible allowing time for the inevitable clean-up before the final ranking list acceptance process.

In its most restricted form the problem may be expressed as, given an assignment matrix of proposals to evaluators, where each proposal has been read by some subset of evaluators, generate a schedule of consensus meetings that uses the fewest number of time periods where the meetings can be held. A maximum of  $T$  time periods exist.

The constraints that must be respected, then, are:

1. A consensus meeting can only take place during one time slot;
2. An evaluator can only be in one consensus meeting during a time slot;
3. If a consensus meeting for a proposal takes place then all of the evaluators assigned to it must be present;
4. No more than  $T$  time slots may be used.

The goal is to minimize the number of time slots used. While the problem in its purest form is similar to the exam timetabling problem, we will see some additional constraints that impinge on it later.

In the following section we describe our first approach to solve the problem exactly by modelling it as an ILP. This will act as a reference point for what will follow later. We then describe a two-phase heuristic that combines tabu search to find an initial colouring with a weighted bipartite graph matching problem on the colouring, and the deficiencies of this model. Section 4 proposes a return to the search for an exact solution to a fundamental sub-problem by comparing two competing ILP formulations. Section 5 concludes the paper.

## 2 An ILP Model

Given  $E = e_{ij}$ , a binary assignment matrix that indicates what evaluators have been assigned to read proposal  $j$ , we can view its transpose  $E^T = P = p_{ij}$  as the matrix that indicates what proposals have been assigned to evaluator  $j$ .

We introduce binary variables  $x_{ij}$  that indicate that the consensus meeting for proposal  $i$  will take place in time slot  $j$ , and  $y_{ij}$  that indicate that evaluator  $i$  is in some meeting during time slot  $j$ ,  $1 \leq j \leq T$ .

Constraint [1](#) above can be implemented by

$$\sum_j x_{ij} = 1 \quad \forall i.$$

Constraint [2](#) says that if an evaluator is assigned to a time slot then s/he must be evaluating exactly one proposal from their allocation and, conversely, if an evaluator is not assigned to a time slot then none of their allocation are being evaluated in this slot:

$$y_{ij} = \sum_k e_{ik} x_{kj} = \sum_k p_{ki} x_{kj} \quad \forall i, j.$$

Constraint [3](#) can be interpreted as meaning ‘meeting for proposal  $i$  happens at time  $j \Rightarrow$  all of the evaluators associated with this proposal are assigned to this slot’. Note that this relation is not ‘ $\Leftrightarrow$ ’ since the same three evaluators may be involved in a different proposal. Also, by virtue of constraint [1](#) an evaluator can only attend one meeting in a time slot. Its implementation is

$$w_i x_{ij} \leq \sum_k p_{ik} y_{kj} = \sum_k e_{ki} y_{kj} \quad \forall i, j$$

where  $w_i = \sum_j p_{ij}$  is the number of evaluators assigned to proposal  $i$ .

The goal is to minimize the number of time slots used. To do this we seek to minimize a new variable  $z \geq \max_j \{j | x_{ij} = 1, \forall i\}$ . That is, if a proposal is assigned to slot  $j$  then  $z \geq j$  and we wish to minimize  $z$ . Equation [5](#) implements this constraint.

Thus the ILP model we solve is

$$\text{minimize } z \tag{1}$$

subject to

$$\sum_j x_{ij} = 1 \quad \forall i \tag{2}$$

$$y_{ij} = \sum_k e_{ik} x_{kj} \quad \forall i, j \tag{3}$$

$$w_i x_{ij} \leq \sum_k e_{ki} y_{kj} \quad \forall i, j \tag{4}$$

$$j x_{ij} \leq z \quad \forall i, j \tag{5}$$

$$x_{ij}, y_{ij} \quad \text{binary.}$$

For a problem instance involving 58 evaluators and 383 proposals using CPLEX 7.0 and running on Linux using an Intel Pentium with a clock speed of 350 MHz and 128 Mb of RAM, the previous model had not terminated after three days of running time.

Of equal concern was the inadequacy of the implemented model. It often arises that an evaluator cannot be present for the entire duration or may arrive late, and



thus not all slots are equally suitable. Further, some evaluators (e.g. vice-chairs) have other duties and it is desirable that their consensus meetings be scheduled as early as possible. (One further constraint that the system was required to deal with was that, on occasion, the entire panel meeting is actually a coalition of smaller panels, with evaluators involved in some or all of these smaller panels. It was desirable that smaller panels were completed as soon as possible, allowing the data entry and ranking list generation to take place for these smaller panels.)

An alternative solution strategy is described below.

### 3 A Refined Model

In addition to constraints [1-4](#), the following refinements are now also considered to address the deficiencies described above.

- R1 No consensus meeting may be scheduled at a time when not all assigned evaluators are present;
- R2 Evaluators with other duties should be scheduled to finish their consensus meeting duties as early as possible;
- R3 Some proposals (for example, those associated with a smaller sub-panel) should be scheduled as early as possible.

Given a set of proposals  $\mathcal{P}$  and a set of evaluators  $\mathcal{E}$  with a slight abuse of notation we define functions  $\mathcal{P} : \mathcal{E} \rightarrow \mathcal{P}^*$ ,  $\mathcal{E} : \mathcal{P} \rightarrow \mathcal{E}^*$  and  $\mathcal{A} : \mathcal{E} \rightarrow \{1, \dots, T\}^*$  which, respectively, identify the set of proposals an evaluator  $e_i \in \mathcal{E}$  has been allocated, the set of evaluators assigned to a proposal and the set of time slots for which evaluator  $e_i$  can be available.

Graph colouring has long been associated with timetabling [4,3,6,13](#) and we adopt this approach here. For the model presented in Section [2](#) we construct a graph  $G = (V_{\mathcal{P}}, E_{\mathcal{P}})$ , where the vertices represent proposals and an edge exists between two vertices if the corresponding proposals have one or more evaluators in common. In any legal vertex colouring, vertices of the same colour may be scheduled together since they are guaranteed to be non-adjacent. In the absence of limits on evaluators availabilities  $P_i$ , the proposals of colour  $i = 1, \dots, C$ , may be scheduled, respectively, in time slots  $t_i, i = 1, \dots, C$ .

Using a tabu-search-based vertex colouring algorithm gives quite satisfactory results on problem instances commensurate with that described earlier. While it is difficult to find good lower bounding strategies for the graph colouring problem in general, the special structure of  $G$  provides us with a good bound in this case: the number of colours that is necessary to colour the graph will be at least the number of slots required by the busiest evaluator. For the cases we have solved it was quite common to find a colouring that required no more slots than the busiest evaluator.

#### 3.1 Restricted Evaluator Availabilities

Restricted availability of one or more evaluators can be accommodated by solving a *maximum cardinality bipartite matching* instance [2](#). We construct the

bipartite graph  $B = (U, W, E)$ , where  $U = \{i | 1 \leq i \leq C\}$  is the set of colour classes and  $W = \{j | 1 \leq j \leq T\}$ . Vertices  $u$  and  $w$  are connected by an edge if and only if every evaluator involved with proposals  $P_u$  is available during time period  $w$ . The neighbourhood of  $u$  is the set of time slots in which all proposals  $P_u$  may be feasibly scheduled.

However, since the colouring phase is unaware of sequencing effects of vertex colourings, problems can arise. For example, two proposals assigned to the same colour class may require evaluators who cannot be present at the panel meeting simultaneously. Accordingly this will result in vertex  $u \in U$  having 0 neighbours, and thus not schedulable. More generally, if two evaluators  $e_1$  and  $e_2$  will be present simultaneously for no more than  $\tau = |\mathcal{A}(e_1) \cap \mathcal{A}(e_2)|$  periods, then we must ensure that the set of vertices are partitioned into colour classes in such a way that no more than  $\tau$  colour classes will contain vertices  $v_1 \neq v_2$  where  $v_1 \in \mathcal{P}(e_1)$  and  $v_2 \in \mathcal{P}(e_2)$ . (Of course the problem is infeasible and without solution if  $e_1$  and  $e_2$  have been allocated more than  $\tau$  of the *same* proposals; when reading the availabilities of evaluators initially it is easy to check for this occurrence.)

Feasible colourings can still be found that adhere to availability restrictions by augmenting the graph. We impose additional edges in  $E_{\mathcal{P}}$  so that all but  $\tau$  edges  $(u, v)$  are connected, where  $u \in \mathcal{P}(e_1) \setminus \mathcal{P}(e_2)$  and  $v \in \mathcal{P}(e_2) \setminus \mathcal{P}(e_1)$ . If  $e_1$  and  $e_2$  are both assigned to  $c = |\mathcal{P}(e_1) \cap \mathcal{P}(e_2)|$  proposals then all but  $\tau' = \tau - c$  must be forced apart at the colouring stage. Note that due to constraints on *other* evaluators, some of these edges may already be present. By selecting up to  $\tau' \leq \min(|\mathcal{P}(e_1) \setminus \mathcal{P}(e_2)|, |\mathcal{P}(e_2) \setminus \mathcal{P}(e_1)|)$  vertices from each set we connect all other pairings with an edge and thus force them apart.

Figure [1](#) illustrates an example. Evaluator  $e_1$  has been assigned proposals  $p_1, p_2, p_3, p_4$  and  $p_5$ , while evaluator  $e_2$  has been assigned proposals  $p_1, p_2, p_6$  and  $p_7$ . Suppose they are only available simultaneously for four time slots. Since they co-evaluate proposals  $p_1$  and  $p_2$ , we must ensure that no more than two more of their shared proposals are scheduled together. Therefore we need to ensure that there are no more than two edges missing from the bipartite graph comprising vertex sets  $\{p_3, p_4, p_5\}$  and  $\{p_6, p_7\}$ . The edge  $(p_5, p_7)$  already exists due to both proposals being evaluated by a common evaluator and, therefore, we impose  $3 \cdot 2 - 1 - 2 = 3$  additional (thick) edges.

The introduction of edges assists in facilitating evaluators when they are highly restricted. However, in the case of evaluators with, for example, no restrictions on their availabilities the above procedure requires insertion of edges unnecessarily. Further, it is not clear how the choice of the imposed edges affects the performance of the tabu-based colouring algorithm either in terms of its run-time or quality of solution; a small number of evaluators with restricted availabilities has been observed to have an effect on the time taken to find a solution and this may be due to the choice of edges introduced. While this may turn out to be an interesting area of research it was also the motivation for us to find an exact algorithm that enforced evaluator availabilities without this indeterminacy. This is discussed further in Section [4](#).

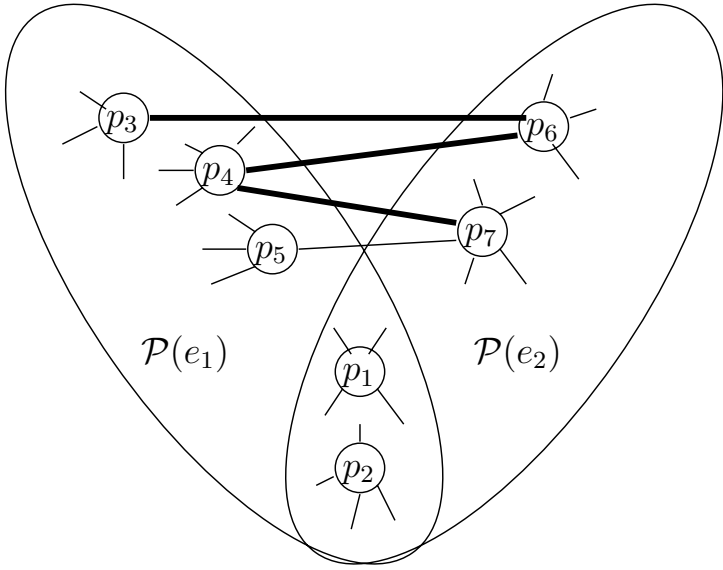


Fig. 1. Imposing additional edges on the workload of two evaluators

### 3.2 Soft Constraints

Constraints **R2** and **R3** are treated differently and with less urgency since they do not affect feasibility. A family of partial solutions is found by colouring the proposals subject to the further restrictions generated by the overlap of evaluators’ availabilities. In a second phase of the algorithm a solution is created by assigning each colour to a time slot and it is in this second phase that secondary constraints, such as biasing a small set of evaluators with other duties to finish early or a set of proposals to be completed early, are introduced.

We build the bipartite graph as described at the start of the previous section but we now add weights to edges. Initially every edge  $(u, v)$  has weight 1 but under certain circumstances these weights may be augmented by the following process: for a colour class  $u$  and its neighbourhood,  $N(u) = \{v_{i_1}, v_{i_2}, \dots, v_{i_m}\}, 1 \leq i_j \leq T, |N(u)| = m$ , a weight or bias  $b^{i_j}, 0 < b < 1$  is added to each such edge. Different biases may be used for **R2** and **R3**.

On this weighted bipartite graph we call a *maximum weighted bipartite matching* algorithm, which has the effect of choosing earlier time slots for a coloured set of proposals.

In the case of constraint **R2** each vice-chair is considered in turn, and the previous process is applied to the colourings in which their proposals appear. Likewise, in order to accommodate constraint **R3**, if a proposal is marked as requiring early completion then it can be thus biased. Funding agency officials have the ability to specify different weightings depending on their priorities.

The system is implemented in Perl and, when run on Linux using an Intel Pentium IV with a clock speed of 2.8 GHz and 512 Mb of memory and using problem instance data of the magnitude discussed earlier, returns a schedule (or indicates that there is an infeasibility) in a few seconds. We have also solved problems along the scale of 110 evaluators and 1,000 proposals in approximately 30 seconds.

It is possible that an evaluation function could be constructed that would bias the colouring phase towards solutions that are more favourable according to the soft criteria. However, separating the problem into two phases allows use of ‘off the shelf’ implementations of tabu search and maximum-weighted bipartite matching algorithms.

## 4 The Range-Colouring Problem

We have seen that the consensus meeting scheduling problem reduces to a graph colouring problem where (due to evaluators’ availabilities) vertices may only be coloured from a prescribed set. This is known in the research literature as the *list colouring* problem. While some theoretical results exist for the problem [11,12,13] few practical algorithms are known to this author, although for special cases such as if the graph is an interval graph some efficient algorithms exist [14].

Due to the unique circumstances of the problem with evaluators usually travelling long distances the set of available time slots for any evaluator,  $e_i$ , is always contiguous,  $\mathcal{A}(e_i) = [t_s, \dots, t_e]$ . Thus, instead of an arbitrary set of colour restrictions being imposed on a vertex  $p$ , the restrictions form an interval of colours  $[c_i, \dots, c_j]$  defined by the intersection of the evaluators’ availabilities,  $\bigcap_{e \in \mathcal{E}(p)} \mathcal{A}(p)$ . For this reason we call this special case of the list colouring problem the *range colouring* problem.

While it is almost certain that heuristic approaches will prove most viable for large-scale problems, the relative absence of any algorithms for the list-colouring problem suggested that we begin with evaluating exact algorithms for the problem. We present later an ILP formulation of the range colouring problem that, we believe, will be the basis of a successful branch-and-cut attack on the problem. It is perhaps testament to the difficulty of the graph colouring problem that so few branch-and-cut papers appear in the literature. Méndez Díaz and Zabala (MDZ) present what they claim to be a successful formulation of the *graph colouring* problem [10] and we use this in order to compare our algorithm.

They base their ILP on the classical formulation of the graph colouring problem as follows:

$$\text{minimize } \sum_{j=1}^n w_j \quad (6)$$

subject to

$$\sum_j x_{ij} = 1 \quad \forall i \quad (7)$$

$$x_{ij} + x_{kj} \leq w_j \quad \forall \{i, k\} \in E, 1 \leq j \leq n \quad (8)$$

$$w_j \leq \sum_{i \in V} x_{ij} \qquad 1 \leq j \leq n \qquad (9)$$

$$w_j \geq w_{j+1} \qquad 1 \leq j \leq n - 1 \qquad (10)$$

$$x_{ij} \qquad \forall i \in V, 1 \leq j \leq n \quad \text{binary}$$

$$w_j \qquad 1 \leq j \leq n \quad \text{binary.}$$

Binary variable  $x_{ij}$  indicates the assignment of colour  $j$  to vertex  $i$ . Binary variable  $w_j$  indicates the use of colour  $j$ . Equality (7) assigns exactly one colour to every vertex and inequality (8) ensures that adjacent vertices are coloured differently. To avoid many identical, symmetric solutions inequalities (9) and (10) help to eliminate such symmetrical solutions. The upper bound of  $n$  on the number of colours used can be replaced by a tighter upper bound, if known.

### 4.1 An Exact Algorithm for Range Colouring

The motivation for our formulation for graph colouring comes from the DAG layering problem [7][8]. The classical DAG layering problem seeks to assign vertices of a DAG to discrete layers such that all edges point downwards where each ‘layer’ is a set of non-adjacent vertices and can thus be coloured similarly. Our MILP formulation searches for an assignment of vertices to horizontal layers, with the vertices on each layer comprising an independent set, without the requirement of edge orientations. The variable  $y[p]$  indicates the horizontal layer (colour) that proposal  $p$  is assigned to. For all edges,  $(p, q) \in E_{\mathcal{P}}$ , we require that  $|y[p] - y[q]| \geq 1$ :

$$\text{minimize } z \qquad (11)$$

subject to

$$1 \leq y[p] \leq C^* \qquad \forall p \in \mathcal{P} \qquad (12)$$

$$y[p] \leq z \qquad \forall p \in \mathcal{P} \qquad (13)$$

$$y[p] - y[q] - C^* \delta_{pq} \geq 1 - C^* \qquad \forall (p, q) \in E_{\mathcal{P}} \qquad (14)$$

$$y[p] - y[q] - C^* \delta_{pq} \leq -1 \qquad \forall (p, q) \in E_{\mathcal{P}} \qquad (15)$$

$$\delta_{pq} \qquad \text{binary.}$$

The parameter  $C^*$  is an upper bound on the number of colours required. Minimizing the number of colours used is achieved by requiring that the ‘colour’ assigned to each proposal is less than  $z$  and minimizing  $z$ . Enforcing a colour separation of at least 1 between adjacent proposals is achieved by inequalities (14) and (15). For this binary indicator variables  $\delta_{pq}$  are required that indicate whether proposal  $p$  is assigned a higher or lower colour than  $q$ . Note that variable  $z$  is a real value and the variables  $y[p]$  are integral. With an appropriate modification of the objective function they may be relaxed also to real-valued variables.

This formulation is more compact than the MDZ one requiring fewer binary variables ( $|E_{\mathcal{P}}|$ ) versus  $(|V_{\mathcal{P}}| + 1) \cdot C^*$ , which may deteriorate to  $(|V_{\mathcal{P}}| + 1) \cdot V_{\mathcal{P}}$ ,

**Table 1.** Comparison of formulations with no evaluator availability restrictions

$ \mathcal{P} $	MDZ	Range
45	0.12	0.94
60	0.13	1.66
75	0.36	4.02
90	1.27	7.41
105	5.31	–

and, asymptotically, the same number of constraints. We conducted a pilot study to compare the two formulations using the generic CPLEX MIP solver on both.

In order to solve the *range colouring* problem small modifications are required in each formulation. In the MDZ formulation we need to replace equality (7) with

$$\sum_{j=t_{i,s}}^{t_{i,e}} x_{ij} = 1$$

where  $t_{i,s}$  is the start time for proposal  $i$  and  $t_{i,e}$  is the end time. In our formulation we must replace Equation (12) with

$$t_{p,s} \leq y[p] \leq t_{p,e} \quad \forall p \in \mathcal{P}$$

where  $t_{p,s}$  indicates the start time of proposal  $p$  and  $t_{p,e}$  its end time.

## 4.2 Experimental Evaluation

The two models described in the previous section were implemented using AMPL and compared. Based on a large allocation of over 650 proposals to over 80 evaluators smaller problems of size 45, 60, 75, 90 and 105 proposals and their associated number of evaluators were created by randomly choosing from the pool of 650 proposals.

Firstly, for each problem size the two model formulations were run with no restrictions on evaluators' availabilities. The number of CPU seconds of running times are reported in Table 1. The optimum number,  $o$ , of slots found was recorded and used for a second set of experiments (to be described below). In many instances completion of the run was sped up by using the known lower bound of the largest allocation to an evaluator as a constraint on the optimum; once this lower bound was reached the solver could exit immediately. On the largest problem the Range Colouring formulation failed to complete in 60 seconds and was terminated, although at the point of termination it had discovered a solution within 1 of the optimal number of slots.

Using the problem sets generated previously we then set to investigate the effectiveness of the two formulations in the presence of restrictions on some evaluators. The MDZ and Range Colouring AMPL formulations were adjusted

**Table 2.** Comparisons of formulations with reduced availabilities

$ \mathcal{P} $	$ \mathcal{P}' $	$S$ (%)	MDZ	Range
45	18	88.6	0.39	0.85
	36	66.7	0.16	6.33
60	20	91.7	4.92	14.74
	43	65.2	3.64	54.50
75	23	89.8	14.49	5.66
	53	74.9	12.57	–
90	19	93.7	23.34	2.19
	50	77.3	16.48	8.06
	60	71.3	15.28	19.06

as discussed at the end of Section 4.1. Although it is evaluators who cause the restriction it is on proposals and when they may be evaluated that their impact is felt. Through selecting a random set of evaluators and reducing their availability we sought to create at least two different scenarios for each problem as described below.

Table 2 below details the results of our second round of experiments. (Introducing restrictions on the 105-proposal problem resulted in problems that did not complete within our 60-second cutoff limit.) The column labelled  $|\mathcal{P}|$ , as with its full availability counterpart, denotes the number of proposals under consideration. The second column, labelled  $|\mathcal{P}'|$ , indicates the number of proposals that can no longer be scheduled in all slots as a result of imposing reductions in availabilities on a set of evaluators. The column labelled  $S$  indicates the percentage of slots which are usable in this scenario. Using  $o$ , the optimum number of slots required in the unconstrained version, it is calculated as

$$S = \frac{|\mathcal{P} \setminus \mathcal{P}'| \times o + \sum_{p' \in \mathcal{P}'} (e(p') - s(p') + 1)}{|\mathcal{P}| \times o} \times 100.$$

That is, assuming an optimum solution of  $o$  again, any proposal that is not affected by missing evaluators may be scheduled in any of the  $o$  slots while each affected proposal  $p'$  may be scheduled in  $e(p') - s(p') + 1$  slots. This is then represented as a percentage of the full  $|\mathcal{P}| \times o$ .

An approximate goal of constructing the alternative scenarios was to constrain the first scenario by removing approximately 10% of the total slots available and to remove approximately 33% in the second scenario. Due to unevenness in the distribution of proposals to evaluators in the 90-proposal case this was difficult to achieve; a third scenario was introduced in this case. The fourth and fifth columns indicate the running times of the two formulations as before.

**Discussion.** For the case of full availabilities of evaluators, as described in Table 1, it is clear that the MDZ formulation is the superior one, with the

Range Colouring formulation not even terminating within the given 60-second time bound on the last problem. This is in spite of it having a better problem formulation profile in terms of number of constraints and variables used – though more complex to understand possibly. It is not known if the cleaner formulation of MDZ makes itself amenable to some problem reduction strategy within AMPL/CPLEX.

For the case of reduced availabilities the situation is less clear. Although MDZ has the upper hand for both constructed scenarios of the 45- and 60-proposal problems, things become more complicated when the two larger problem sizes are considered. When the number of usable slots ( $S$ ) is reduced by approximately 10% (and more for the 90-proposal case) the Range Colouring completes before MDZ. This is somewhat counter-intuitive since MDZ is the winner for both problem sizes in both the more constrained and the less constrained scenarios (see Tables 2 and 1, respectively). This is an aspect that requires further investigation.

Good lower and upper bounds are well known to assist in the speedy running of ILPs. We have remarked previously on how the largest number of proposals allocated to any evaluator provides a very good lower bound on solution quality. We did not spend any effort in devising upper bound heuristics but nonetheless we have observed particular sensitivity of both models to the upper bound used. Both models explicitly use the upper bound though a balance appears to be necessary between using a small upper bound that tightens inequalities and a larger bound that gives the ILP solver its all-important first feasible solution.

It should be pointed out that a further advantage of the MDZ formulation lies in its ability to be modified to forbid *any* set of slots for the scheduling of a proposal. For our purposes this was not necessary; however, in the most general form of restricted colouring this will be the case. Thus, if the modifications of the MDZ model do not seriously hamper its performance this would appear to be a more general and robust model.

Nonetheless, our experiments suggest that the new formulation has a role to play. It appears to be particularly suited to scenarios where there is some – but not a large – impact on the availabilities of evaluators. In view of its similarities to the DAG layering problem an analysis of its polyhedral structure may be worthwhile. We have already identified some basic clique-like inequalities and identifying others may make a branch-and-cut solution practicable.

## 5 Conclusions

In this paper we have described an effective two-phase heuristic procedure for scheduling a set of interrelated meetings that has similarities to the exam timetabling problem. We have also proposed and discussed two ILP formulations that provide exact solutions to the problem. The ILPs can be used to solve the special case of the list colouring problem that we call the range colouring problem.

Many aspects of both topics investigated here are worthy of further consideration. When imposing additional edges in order to force separate colourings of proposals the method of choosing amongst those proposals that could be forcibly



separated is worthy of further investigation, we believe. Also, clearly there is an interaction between the two constraints and the choice of  $b$  for each type of soft constraint and this is an area which can be investigated further.

The problem has been decomposed into a graph colouring subproblem and a matching problem. While the latter is an exact solution, the former finds a heuristically generated colouring. Further, by separating the problem in this manner and ignoring the soft constraints initially we may lose opportunities for finding solutions that are more satisfactory with respect to the soft constraints.

We have compared two formulations of a special case of the list colouring problem that we call the *range colouring* problem. Although there are some anomalies that should be investigated further the MDZ formulation would appear to be the one to be recommended overall. In view of its similarities to the DAG layering formulation it may be worthwhile, nonetheless, to investigate the structure of the Range Colouring polytope.

Prior to the introduction of this system consensus meetings took place in a haphazard, ad hoc fashion, with evaluators wasting much effort searching out their associates in order to discuss a proposal. According to one official, for the scale of problem instance we have discussed in Section 2 the system has resulted in panel meetings being completed a day sooner in the week than heretofore.

*Acknowledgements.* The author acknowledges gratefully the comments of the anonymous referees.

## References

1. Akbari, S., Fana, H.-R.: Some relations among term rank, clique number and list chromatic number of a graph. *Discrete Mathematics* 306, 3078–3082 (2006)
2. Alt, H., Blum, N., Mehlhorn, K., Paul, M.: Computing a maximum cardinality matching in a bipartite graph in time  $o(n^{1.5} \sqrt{m/\log n})$ . *Information Processing Letters* 37, 237–240 (1991)
3. Burke, E.K., de Werra, D., Kingston, J.: Applications to timetabling. In: *Handbook of Graph Theory*, ch. 5.6, Chapman and Hall/CRC Press, London (2004)
4. Burke, E.K., Jackson, K.S., Kingston, J.H., Weare, R.F.: Automated timetabling: The state of the art. *The Computer Journal* 40, 565–571 (1997)
5. Cook, W.D., Golany, B., Penn, M., Raviv, T.: Creating a consensus ranking of proposals from reviewers' partial ordinal rankings. *Computers and Operations Research* 34, 954–965 (2007)
6. Friden, C., Hertz, A., de Werra, D.: STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing* 42, 35–44 (1989)
7. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.P.: A technique for drawing directed graphs. *IEEE Transactions on Software Engineering* 19, 214–230 (1993)
8. Healy, P., Nikolov, N.S.: How to layer a directed acyclic graph. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) *GD 2001*. LNCS, vol. 2265, pp. 16–30. Springer, Heidelberg (2002)
9. Kratochvíl, J., Tuza, Z.: Algorithmic complexity of list colorings. *Discrete Applied Mathematics* 50, 297–302 (1994)

10. Méndez-Díaz, I., Zabala, P.: A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics* 154, 826–847 (2006)
11. Tuza, Z.: Graph colorings with local constraints. *Discussiones Mathematicae, Graph Theory* 17, 161–228 (1997)
12. Vizing, V.G.: Coloring the vertices of a graph in prescribed colors (in Russian). *Metody Diskretnogo Analiza v Teorii Kodov i Schem* 29, 3–10 (1976)
13. Wood, D.C.: A technique for coloring a graph applicable to large-scale timetabling problems. *The Computer Journal* 12, 317–322 (1969)
14. Zeitlhofer, T., Wess, B.: List-coloring of interval graphs with application to register assignment for heterogeneous register-set architectures. *Signal Processing* 83, 1411–1425 (2003)

# **Sports Timetabling**

# Constructive Algorithms for the Constant Distance Traveling Tournament Problem

Nobutomo Fujiwara<sup>1</sup>, Shinji Imahori<sup>1</sup>, Tomomi Matsui<sup>2</sup>,  
and Ryuhei Miyashiro<sup>3</sup>

<sup>1</sup> Graduate School of Information Science and Technology,  
The University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan  
nobutomo@simplex.t.u-tokyo.ac.jp

imahori@mist.i.u-tokyo.ac.jp

<sup>2</sup> Faculty of Science and Engineering, Chuo University,  
Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan

matsui@ise.chuo-u.ac.jp

<sup>3</sup> Institute of Symbiotic Science and Technology,  
Tokyo University of Agriculture and Technology,  
Naka-cho, Koganei, Tokyo 184-8588, Japan

r-miya@cc.tuat.ac.jp

**Abstract.** The traveling tournament problem considers scheduling round-robin tournaments that minimize traveling distance, which is an important issue in sports scheduling. Various studies on the traveling tournament problem have appeared in recent years, and there are some variants of this problem. In this paper, we deal with the constant distance traveling tournament problem, which is a special class of the traveling tournament problem. This variant is essentially equivalent to the problem of ‘maximizing breaks’ and that of ‘minimizing breaks’, which is another significant objective in sports scheduling. We propose a lower bound of the optimal value of the constant distance traveling tournament problem, and two constructive algorithms that produce feasible solutions whose objective values are close to the proposed lower bound. For some size of instances, one of our algorithms yields optimal solutions.

## 1 Introduction

In scheduling of round-robin tournaments, there are two major objectives [9], ‘minimizing breaks’ and ‘minimizing traveling distance’. The former is to improve quality of a tournament by decreasing the number of breaks (consecutive games both held at away or both held at home); the latter aims to reduce traveling costs of teams by minimizing traveling distance. Many papers that concern minimizing traveling distance have been published so far. The traveling tournament problem (TTP), established by Easton et al. [3], is a well-known benchmark problem that abstracts the concept of minimizing traveling distance. The constant distance traveling tournament problem (CDTTP), introduced by Urrutia and Ribeiro [13], is a variant of TTP, in which all distance between

pairs of teams are one. It is known that maximizing breaks gives an approximate solution for minimizing traveling distance [11]. In particular, in CDTTP maximizing breaks is not approximation but direct transformation of minimizing distance [13]. Moreover, it was also shown that the problem of maximizing breaks is essentially equivalent to that of minimizing breaks [6], which is another significant subject in sports scheduling.

Most of the best upper bounds of CDTTP (and TTP) are obtained by meta-heuristic algorithms [11, 12, 14]. In contrast, it is difficult to obtain good lower bounds for CDTTP; a logic based Benders decomposition approach [7] was used to obtain lower bounds of instances of up to 16 teams, and lower bounds of larger instances are not known so far. In this paper, we propose a new lower bound for CDTTP, and two algorithms that produce feasible solutions whose objective values are close to the proposed lower bound. For some size of instances, one of our algorithms yields optimal solutions.

## 2 Problem

In this section, we introduce some terminology and definitions, and then describe the constant distance traveling tournament problem (CDTTP). For more discussions on CDTTP and its variations, see [7, 13].

We are given a set of teams  $T = \{1, 2, \dots, n\}$  where  $n$  is an even number, and each team has its home venue. A game is specified by an ordered pair of teams. A double round-robin tournament is a set of games in which every team plays every other team once at its home venue and once at away (i.e., at the venue of the opponent); hence, exactly  $2(n - 1)$  slots are required to complete a double round-robin tournament.

Each team stays its home venue before a tournament, and then travels to play games at the chosen venues. After a tournament, each team goes back to its home venue. In a tournament, the number of trips of a team is defined by the number of moves of the team between venues. We note that, when a team plays two consecutive away games, the team goes directly from the venue of the first opponent to the other, without returning to its home. Consecutive away games for a team constitute a *road trip*; consecutive home games are a *home stand*. The length of a road trip/home stand is the number of opponents playing against in the road trip/home stand, respectively. The constant distance traveling tournament problem is defined as follows.

### Constant Distance Traveling Tournament Problem

Input: the number of teams,  $n$ ;

Output: a double round-robin tournament of  $n$  teams such that

1. the length of any home stand and that of any road trip are at most three;
2. no repeaters (A at B immediately followed by B at A is prohibited);
3. the total number of trips taken by teams is minimized.

In the rest of this paper, a double round-robin tournament satisfying the above conditions 1 and 2 is called a *feasible tournament*.

We note that CDTTP is a special class of the original traveling tournament problem [3] such that the distance between any pair of home venues is equal to one.

Given a feasible tournament, it is said that a team has a *break* at slot  $s$  if it has two consecutive home games or two consecutive away games in slots  $s - 1$  and  $s$ . We also say that a team has a *home break* (resp., *away break*) at a game if both of the game and the previous game are at home (resp., away). In a feasible tournament  $S$ , the total number of breaks  $B(S)$  is defined as the sum of the number of breaks of all the teams. As for the number of trips and the number of breaks, the following lemma is known.

**Lemma 1 (Urrutia and Ribeiro [13]).** *Let  $S$  be a feasible tournament for CDTTP. The total number of trips  $D(S)$  and the total number of breaks  $B(S)$  have the following relationship:*

$$D(S) = 2n(n - 1) - B(S)/2.$$

Thus, maximizing the total number of breaks  $B(S)$  is essentially equivalent to minimizing the total number of trips  $D(S)$ . Moreover, maximizing the number of breaks is essentially equivalent to minimizing the number of breaks [6] if the constraint on the length of a road trip/home stand is not supposed.

### 3 Lower Bound

In this section, we propose a new lower bound for CDTTP. In previous researches on CDTTP, the logic-based Benders decomposition approach [7] was used to obtain lower bounds of instances of up to 16 teams, and lower bounds of larger instances are not known so far. Our analysis produces a lower bound for every size of the CDTTP instance. We note that Urrutia and Ribeiro proposed a lower bound for every instance of the mirrored CDTTP [13].

A home-away pattern of a team is a vector of  $2(n - 1)$  elements in which each element is ‘H’ or ‘A’, where H means ‘at home’ and A ‘away’. For example, a home-away pattern HHAAAH of a team means that the team plays games at its home venue in slots 1, 2 and 6, while it plays games at venues of opponents in slots 3, 4 and 5.

**Theorem 1.** *For every feasible tournament of  $n$  teams, the number of trips is greater than or equal to  $LB(n)$  defined by*

$$LB(n) \stackrel{\text{def.}}{=} \begin{cases} (4/3)n^2 - n & (n \equiv 0 \pmod 3), \\ (4/3)n^2 - (5/6)n - 1 & (n \equiv 1 \pmod 3), \\ (4/3)n^2 - (2/3)n & (n \equiv 2 \pmod 3). \end{cases}$$

**Proof.** First, consider the case that  $n \equiv 0 \pmod 3$ . The number of slots satisfies  $2(n - 1) \equiv 1 \pmod 3$ . Since no team can have three breaks in any consecutive three slots, each team can have at most two breaks in each consecutive three slots of slots 2, 3, 4, slots 5, 6, 7, . . . , slots  $2n - 4, 2n - 3, 2n - 2$ . In addition, no

team can have a break at slot 1. Thus, the number of breaks for each team is less than or equal to  $(2/3)(2(n-1) - 1) = (4/3)n - 2$ . The total number of breaks of every feasible tournament is at most

$$n((4/3)n - 2) = (4/3)n^2 - 2n.$$

Hence, the total number of trips of every feasible tournament is greater than or equal to

$$2n(n-1) - (1/2)((4/3)n^2 - 2n) = (4/3)n^2 - n.$$

Next, consider the case that  $n \equiv 1 \pmod{3}$ . The number of slots satisfies  $2(n-1) \equiv 0 \pmod{3}$ . No team can have three breaks in consecutive three slots, and hence each team has at most  $(2/3)(2(n-1)) = (4/3)(n-1)$  breaks. Moreover, there are only two home-away patterns including  $(4/3)(n-1)$  breaks; that is, HHHAAAHHH  $\cdots$  HHHAAA and AAHHHAAA  $\cdots$  AAHHH. Since no pairs of teams have the same home-away pattern [2], at most two teams are possible to have  $(4/3)(n-1)$  breaks, and the other teams can have at most  $(4/3)(n-1) - 1$  breaks. Thus, the total number of breaks of every feasible tournament is less than or equal to

$$2((4/3)(n-1)) + (n-2)((4/3)(n-1) - 1) = (4/3)n^2 - (7/3)n + 2.$$

Consequently, the total number of trips of every feasible tournament is greater than or equal to

$$2n(n-1) - (1/2)((4/3)n^2 - (7/3)n + 2) = (4/3)n^2 - (5/6)n - 1.$$

Finally, consider the case that  $n \equiv 2 \pmod{3}$ . The number of home games and that of away games for each team satisfy  $n-1 \equiv 1 \pmod{3}$ . Each team has breaks neither at its first home game nor at its first away game. In addition, no teams have three breaks in consecutive three games. Thus, each team has at most  $2(2/3)((n-1) - 1) = (4/3)(n-2)$  breaks. The total number of breaks of every feasible tournament is less than or equal to

$$n((4/3)(n-2)) = (4/3)n^2 - (8/3)n.$$

Hence, the total number of trips of every feasible tournament is greater than or equal to

$$2n(n-1) - (1/2)((4/3)n^2 - (8/3)n) = (4/3)n^2 - (2/3)n.$$

□

## 4 Algorithms

In this section, for constructing good feasible tournaments we propose two algorithms named the Modified Circle Method and the Minimum Break Method. The proposed algorithms are constructive ones, whereas most of the previous upper bounds of CDTTP (and TTP) are obtained by metaheuristic algorithms [1,2,14]. Some constructive algorithms for TTP were proposed in papers [8,10]. Both of our algorithms work similarly: construct specific single round-robin tournaments, and then modify them into double round-robin tournaments.

### 4.1 Modified Circle Method

In this section, we propose an algorithm named *Modified Circle Method* (MCM for short). First, we describe the procedure of MCM, and then show: in Lemma 2 feasibility of the tournament obtained by MCM; in Lemma 3 the number of breaks of the tournament; in Theorem 2 the number of trips of the tournament.

Denote the set of teams by  $T = \{1, 2, \dots, n\}$ . We introduce a directed graph  $G^e = (T, A^e)$  with a vertex set  $T$  and a set of mutually disjoint directed edges

$$A^e \stackrel{\text{def.}}{=} \{(j, n + 1 - j) : \lceil j/3 \rceil \text{ is even, } 1 \leq j \leq n/2\} \cup \{(n + 1 - j, j) : \lceil j/3 \rceil \text{ is odd, } 1 \leq j \leq n/2\}.$$

Let  $G^o = (T, A^o)$  be a directed graph obtained from  $G^e$  by reversing the direction of the edge between 1 and  $n$ . For each  $s \in \{1, 2, \dots, n - 1\}$ , we define a permutation  $\pi^s$  by  $(\pi^s(1), \pi^s(2), \dots, \pi^s(n)) = (s, s + 1, \dots, n - 1, 1, 2, \dots, s - 1, n)$ . For any permutation  $\pi$  on  $T$ ,  $G^e(\pi)$  (resp.,  $G^o(\pi)$ ) denotes the set of  $n/2$  games satisfying that every directed edge  $(u, v) \in A^e$  (resp.,  $A^o$ ) corresponds to a game between  $\pi(u)$  and  $\pi(v)$  held at the home venue of  $\pi(v)$ .

Consider the case that  $n \equiv 0 \pmod 3$ . Let  $X$  be a single round-robin tournament satisfying that games in slot  $s$  are defined by  $G^o(\pi^s)$  if  $s \in \{1, 2, 3\} \pmod 6$ , and by  $G^e(\pi^s)$  if  $s \in \{4, 5, 0\} \pmod 6$ . Figure 1 shows the games of the first four slots in  $X$  when  $n = 18$ . For each  $i \in \{1, 2, \dots, n/3 - 1\}$ , we denote a partial schedule of  $X$  consisting of a sequence of three slots  $(3i - 2, 3i - 1, 3i)$  by  $X_i$ . In addition, we denote the partial schedule of  $X$  consisting of two slots  $(n - 2, n - 1)$  by  $X_{n/3}$ . Now we construct a double round-robin tournament  $Y$  by concatenating these partial schedules as follows:  $Y = (X_1, \overline{X_1}, \overline{X_2}, X_2, X_3, \overline{X_3}, \overline{X_4}, X_4, X_5, \dots, \overline{X_{\frac{n}{3}-1}}, \overline{X_{\frac{n}{3}}}, X_{\frac{n}{3}})$ , where  $\overline{X_i}$  is the partial schedule obtained from  $X_i$  by reversing all venues.

Consider the case that  $n \equiv 1 \pmod 3$ . We construct a single round-robin tournament  $X$  with the same method as above. For each  $i \in \{1, 2, \dots, (n - 1)/3\}$ , we denote a partial schedule of  $X$  consisting of a sequence of three slots  $(3i - 2, 3i - 1, 3i)$  by  $X_i$ . We construct a double round-robin tournament  $Y$  by concatenating the partial schedules as follows:  $Y = (X_1, \overline{X_1}, \overline{X_2}, X_2, X_3, \dots, X_{\frac{n-1}{3}}, \overline{X_{\frac{n-1}{3}}})$ .

Consider the case that  $n \equiv 2 \pmod 3$ . Let  $\tilde{G}^e$  (resp.,  $\tilde{G}^o$ ) be a directed graph obtained from  $G^e$  (resp.,  $G^o$ ) by reversing the direction of the edge between  $n/2 - 1$  and  $n/2 + 2$ . We construct a single round-robin tournament  $X$  as well as the above cases using directed graphs  $\tilde{G}^e$  and  $\tilde{G}^o$ . For each  $i \in \{2, 3, \dots, (n - 2)/3\}$ , we denote a partial schedule of  $X$  consisting of a sequence of three slots  $(3i - 3, 3i - 2, 3i - 1)$  by  $X_i$ . We denote the partial schedules of  $X$  consisting of two slots  $(1, 2)$  by  $X_1$  and two slots  $(n - 2, n - 1)$  by  $X_{(n+1)/3}$ . We construct a double round-robin tournament  $Y$  by concatenating these partial schedules as follows:  $Y = (X_1, \overline{X_1}, \overline{X_2}, X_2, X_3, \dots, X_{\frac{n+1}{3}}, \overline{X_{\frac{n+1}{3}}})$ .

For all the cases, the single round-robin tournament  $X$  has neither partial home-away patterns HHHH, AAAA, HAH nor AHA; that is, each team has at



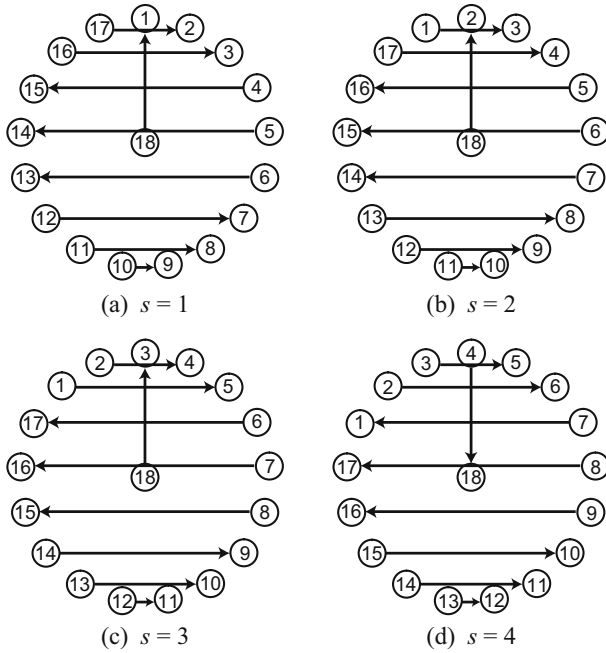


Fig. 1. Games of the first four slots in  $X$ , where  $n = 18$

most three consecutive home/away games in a single round-robin tournament  $X$ . The time complexity to construct  $X$  and  $Y$  is  $O(n^2)$ : i.e., MCM runs in linear time in the output.

**Lemma 2.** *The tournament  $Y$  is a feasible double round-robin tournament.*

**Proof.** It is clear that  $Y$  is a double round-robin tournament. In addition, there are no repeaters in the tournament  $Y$ ; when the game A at B appears in slot  $s$ , the game B at A appears in slot  $s - 3, s + 3, s - 2$  or  $s + 2$ .

We show that the length of any home stand and that of any road trip in  $Y$  is at most three. First, consider partial schedules  $(X_i, \bar{X}_i)$  and  $(\bar{X}_i, X_i)$ . Four consecutive games in these partial schedules must have both of home and away games since they include two games of the same pair of teams. Thus, the maximum length of home stand/road trip in these partial schedules is at most three. Next, we consider partial schedules  $(X_i, X_{i+1})$  and  $(\bar{X}_i, \bar{X}_{i+1})$ . Each team has at most three consecutive home/away games in single round-robin tournaments  $X$  and  $\bar{X}$ , where  $\bar{X}$  is the single round-robin tournament obtained from  $X$  by reversing all venues. Thus, the maximum length of home stand/road trip in these partial schedules is also at most three. Hence, the length of any home stand and that of any road trip in  $Y$  are at most three.  $\square$

**Lemma 3.** *The number of breaks of the double round-robin tournament  $Y$  obtained by MCM satisfies that*

$$B(Y) = \begin{cases} (4/3)n^2 - (8/3)n + 2 & (n \equiv 0 \pmod{3}), \\ (4/3)n^2 - 3n + (8/3) & (n \equiv 1 \pmod{3}), \\ (4/3)n^2 - (13/3)n + (10/3) & (n \equiv 2 \pmod{3}). \end{cases}$$

**Proof.** We first consider the second slot of a partial schedule  $X_i$  consisting of three slots. If a team has a break at the second slot of  $X_i$  in  $X$ , this team has breaks both at the second slots of  $X_i$  and at  $\overline{X}_i$  in  $Y$ ; conversely, if a team has no break at the second slot of  $X_i$  in  $X$ , this team has no breaks both at the second slots of  $X_i$  and at  $\overline{X}_i$  in  $Y$ . These properties also hold both at the third slot of  $X_i$  consisting of three slots and at the second slot of  $X_i$  consisting of two slots.

We then consider the first slot of a partial schedule  $X_i$ . To count the number of breaks at the first slots of  $X_i$  and  $\overline{X}_i$  in  $Y$ , we check the following: (1) a team has a break or not at the first slot of  $X_i$  in  $X$ , and (2) a team has different type games (i.e., home game/away game) at the first and last slots of  $X_i$  or not. The number of positive answers for these questions is equal to the number of breaks at the first slots of  $X_i$  and  $\overline{X}_i$  in  $Y$ .

Now, we sum up the number of breaks in  $Y$ . We first consider the case that  $n \equiv 0 \pmod{3}$ . Teams 1,  $u$  ( $u \equiv 0 \pmod{3}$ ) and  $n - 1$  have  $(4/3)n - 2$  breaks each; the other teams have  $(4/3)n - 3$  breaks each. In total, we have  $(4/3)n^2 - (8/3)n + 2$  breaks in the double round-robin tournament  $Y$ .

We then consider the case that  $n \equiv 1 \pmod{3}$ . Team  $n$  has  $(4/3)n - (4/3)$  breaks. Teams  $u$  ( $u \equiv 0 \pmod{3}$ ,  $u < n/2$ ),  $n/2$  and  $v$  ( $v \equiv 1 \pmod{3}$ ,  $n/2 < v < n$ ) have  $(4/3)n - (7/3)$  breaks each. The other teams have  $(4/3)n - (10/3)$  breaks each. Thus, the total number of breaks in  $Y$  is  $(4/3)n^2 - 3n + (8/3)$ .

We finally consider the case that  $n \equiv 2 \pmod{3}$ . Team  $n$  has  $(4/3)n - (8/3)$  breaks. Teams 1,  $u$  ( $u \equiv 0 \pmod{3}$ ,  $u < n/2 - 2$ ),  $n/2 - 2$ ,  $n/2$ ,  $n/2 + 2$  and  $v$  ( $v \equiv 2 \pmod{3}$ ,  $n/2 + 2 < v < n$ ) have  $(4/3)n - (11/3)$  breaks each. The other teams have  $(4/3)n - (14/3)$  breaks each. Hence, the total number of breaks in  $Y$  is  $(4/3)n^2 - (13/3)n + (10/3)$ .  $\square$

Let the total number of trips of  $Y$  be  $D(Y)$ . Using Theorem 1 and Lemmas 1, 2 and 3, we have the following theorem for the Modified Circle Method.

**Theorem 2.** *The Modified Circle Method produces the feasible double round-robin tournament  $Y$  such that*

$$D(Y) = \begin{cases} (4/3)n^2 - (2/3)n - 1 = \text{LB}(n) + (1/3)n - 1 & (n \equiv 0 \pmod{3}), \\ (4/3)n^2 - (1/2)n - 4/3 = \text{LB}(n) + (1/3)n - 1/3 & (n \equiv 1 \pmod{3}), \\ (4/3)n^2 + (1/6)n - 5/3 = \text{LB}(n) + (5/6)n - 5/3 & (n \equiv 2 \pmod{3}). \end{cases}$$

## 4.2 Minimum Break Method

In this section, we propose an algorithm named *Minimum Break Method* (MBM for short). Before the explanation of MBM, see the following lemmas as for the number of breaks in a single round-robin tournament.

**Lemma 4 (de Werra [2]).** *For any single round-robin tournament of  $n$  teams, the number of breaks is at least  $n - 2$ . There exists a single round-robin tournament that has  $n - 2$  breaks for any even  $n$ .*

**Lemma 5 (Miyashiro and Matsui [6]).** *For any single round-robin tournament of  $n$  teams, the number of breaks is at most  $n^2 - 3n + 2$ . There exists a single round-robin tournament that has  $n^2 - 3n + 2$  breaks for any even  $n$ .*

As is the case with MCM proposed in Section 4.1, we first construct specific single round-robin tournaments, and modify them into double round-robin tournaments. Let  $X$  be a single round-robin tournament satisfying the following conditions:

- (C1) the number of breaks  $B(X)$  is equal to  $n - 2$ ;
- (C2) if  $n \in \{0, 1\} \pmod 3$ , no teams have breaks at each slot  $s \in \{1, 4\} \pmod 6$ ;  
if  $n \equiv 2 \pmod 3$ , no teams have breaks at each slot  $s \in \{0, 3\} \pmod 6$ .

As denoted in Lemma 4, any single round-robin tournament must have at least  $n - 2$  breaks; hence,  $X$  is a tournament with the minimum number of breaks. Here, we have two open problems: (1) such single round-robin tournament  $X$  exists or not, and (2) if  $X$  exists, an efficient algorithm to construct  $X$  exists or not. We conjecture that for every even  $n$  there is a single round-robin tournament  $X$  that satisfies Conditions (C1) and (C2).

In order to obtain a single round-robin tournament satisfying Conditions (C1) and (C2), we adopt the following strategy. We first replace Condition (C2) by a stronger condition:

- (C2') if  $n \in \{0, 1\} \pmod 3$ , then no teams have breaks at each slot  $s \in \{1, 2, 4\} \pmod 6$ , and exactly two teams have breaks at each slot  $s \in \{0, 3, 5\} \pmod 6$ ;  
if  $n \equiv 2 \pmod 3$ , then no teams have breaks at each slot  $s \in \{0, 1, 3\} \pmod 6$ , and exactly two teams have breaks at each slot  $s \in \{2, 4, 5\} \pmod 6$ .

If a tournament satisfies Conditions (C1) and (C2'), it also satisfies Conditions (C1) and (C2). (Here we note that (C1) is implied by (C2') and thus (C1) may be deleted.) For any even number  $n \leq 50$ , we have obtained single round-robin tournaments satisfying Conditions (C1) and (C2') by solving integer programming problems using ILOG CPLEX 9.0 [4]. (We also conjecture that for any even  $n$  there exists single round-robin tournaments satisfying Conditions (C2'). This conjecture corresponds to the conjecture proposed in [5].)

Now we construct another single round-robin tournament, say  $X'$ , from  $X$  by reversing venues for each even slot. The tournament  $X'$  satisfies that exactly two teams have  $n - 2$  breaks and other teams have  $n - 3$  breaks; hence, as denoted in Lemma 5,  $X'$  is a single round-robin tournament with the maximum number of breaks. Moreover, every team in  $X'$  has a break at each slot  $s$  such that: if  $n \in \{0, 1\} \pmod 3$ ,  $s \equiv 1 \pmod 3$  except for  $s = 1$ ; if  $n \equiv 2 \pmod 3$ ,  $s \equiv 0 \pmod 3$ .

We then explain how to construct a double round-robin tournament using  $X'$ .

If  $n \equiv 0 \pmod 3$ , we denote a partial schedule of  $X'$  consisting of a sequence of three slots  $(3i - 2, 3i - 1, 3i)$  by  $X'_i$  for each  $i \in \{1, 2, \dots, (n/3) - 1\}$ , and denote the partial schedule of  $X'$  consisting of two slots  $(n - 2, n - 1)$  by  $X'_{n/3}$ .

If  $n \equiv 1 \pmod 3$ , for each  $i \in \{1, 2, \dots, (n - 1)/3\}$ , we denote a partial schedule of  $X'$  consisting of a sequence of three slots  $(3i - 2, 3i - 1, 3i)$  by  $X'_i$ .

If  $n \equiv 2 \pmod 3$ , we denote a partial schedule of  $X'$  consisting of a sequence of three slots  $(3i - 3, 3i - 2, 3i - 1)$  by  $X'_i$  for each  $i \in \{2, 3, \dots, (n - 2)/3\}$ , and denote the partial schedules of  $X'$  consisting of two slots  $(1, 2)$  by  $X'_1$  and two slots  $(n - 2, n - 1)$  by  $X'_{(n+1)/3}$ .

Now we construct a double round-robin tournament  $Y'$  by concatenating these partial schedules as follows:  $Y' = (X'_1, \overline{X'_1}, X'_2, \overline{X'_2}, X'_3, \overline{X'_3}, \dots)$ , where  $\overline{X'_i}$  is a partial schedule obtained from  $X'_i$  by reversing all venues.

**Lemma 6.** *The tournament  $Y'$  is a feasible double round-robin tournament.*

**Proof.** It is clear that  $Y'$  is a double round-robin tournament. No repeaters appear in the tournament  $Y'$ ; when the game A at B appears in slot  $s$ , the game B at A appears in slot  $s - 3, s + 3, s - 2$  or  $s + 2$ .

We then show that the length of any home stand and that of any road trip is at most three. Consider a partial schedule  $(X'_i, \overline{X'_i})$  of six or four slots: four consecutive games in this partial schedule must have both of home and away games since they include two games of the same pair of teams. Next, consider a partial schedule of six or five slots  $(\overline{X'_i}, X'_{i+1})$ : no teams have breaks at the first slot of  $X'_{i+1}$  in the tournament  $Y'$ , thus four or more consecutive home/away games do not appear. Consequently, the length of any home stand and that of any road trip is at most three. □

**Lemma 7.** *The number of breaks of the double round-robin tournament  $Y'$  is*

$$B(Y') = \begin{cases} (4/3)n^2 - 3n + 2 & (n \equiv 0 \pmod 3), \\ (4/3)n^2 - (7/3)n + 2 & (n \equiv 1 \pmod 3), \\ (4/3)n^2 - (11/3)n + 2 & (n \equiv 2 \pmod 3). \end{cases}$$

**Proof.** To prove this lemma, we use the following four properties on  $X'_i$ :

1. all teams have breaks at the first slot of each  $X'_i$  in  $X'$  except for  $X'_1$ ;
2. no teams have breaks at the first slot of each  $X'_i$  in  $Y'$ ;
3. each team has one or two breaks in the second and third slots of each  $X'_i$  consisting of three slots;
4. each team has zero or one break at the second slot of each  $X'_i$  consisting of two slots.

Consider a partial schedule  $X'_i$  of three slots. If a team has two breaks at the second and third slots of  $X'_i$  (i.e., HHH or AAA), this team has four breaks in a partial schedule  $(X'_i, \overline{X'_i})$  in  $Y'$  (i.e., HHHAAA or AAAHHH). If a team has just one break in the second and third slots of  $X'_i$  (i.e., HHA, HAA, AAH or AHH),

this team has three breaks in a partial schedule  $(X'_i, \overline{X'_i})$  in  $Y'$  (i.e., HHAAAH, HAAAHH, AAHHHA or AHHHAA).

Consider a partial schedule  $X'_i$  of two slots. If a team has a break at the second slot of  $X'_i$  (i.e., HH or AA), this team has two breaks in a partial schedule  $(X'_i, \overline{X'_i})$  in  $Y'$  (i.e., HHAA or AAHH). If a team has no break at the second slot of  $X'_i$  (i.e., HA or AH), this team has one break in a partial schedule  $(X'_i, \overline{X'_i})$  in  $Y'$  (i.e., HAAH or AHHA).

Based on the above observation, we have the following relationship:

$$\begin{aligned} (\text{number of breaks in } Y') &= (\text{number of partial schedules of three slots in } X') \\ &\quad + (\text{number of breaks in } X') + 1. \end{aligned}$$

Using this formula, we obtain  $B(Y')$ , i.e., the total number of breaks of  $Y'$ .  $\square$

From Theorem 1 and Lemmas 6 and 7, we have the following theorem for the Minimum Break Method.

**Theorem 3.** *If there is a single round-robin tournament satisfying Conditions (C1) and (C2), the Minimum Break Method produces a feasible double round-robin tournament  $Y'$  such that*

$$D(Y') = \begin{cases} (4/3)n^2 - (1/2)n - 1 = \text{LB}(n) + (1/2)n - 1 & (n \equiv 0 \pmod 3), \\ (4/3)n^2 - (5/6)n - 1 = \text{LB}(n) & (n \equiv 1 \pmod 3), \\ (4/3)n^2 - (1/6)n - 1 = \text{LB}(n) + (1/2)n - 1 & (n \equiv 2 \pmod 3). \end{cases}$$

Note that, as mentioned before, we found single round-robin tournaments satisfying Conditions (C1) and (C2) for  $n \leq 50$ . Thus, using MBM with those single round-robin tournaments, we obtained feasible double round-robin tournaments  $Y'$  up to 50 teams.

## 5 Results

In this section, we summarize our results on CDTTP. For instances of  $n \equiv 0 \pmod 3$ , MCM gives better solutions compared to MBM. In contrast, for instances of  $n \in \{1, 2\} \pmod 3$ , MBM performs better though it needs single round-robin tournaments satisfying Conditions (C1) and (C2). Moreover, when  $n \equiv 1 \pmod 3$ , MBM yields a solution that attains the proposed lower bound  $\text{LB}(n)$ . We obtained single round-robin tournaments satisfying Conditions (C1) and (C2) up to 50 teams. Table 1 shows the results for  $16 \leq n \leq 50$ : for  $n = 16, 22, 28, 34, 40, 46$ , MBM gave optimal solutions; for  $n = 20$ , our lower bound showed that the best solution previously known is optimal.

Note that MCM works for every even  $n$ , and its time complexity is  $O(n^2)$ . Hence, from Theorems 1 and 2, MCM is a polynomial time algorithm that generates asymptotic optimal solutions for CDTTP.

**Table 1.** Results for  $16 \leq n \leq 50$ 

$n$	LB( $n$ )	MCM	MBM	Old
16	327	332	*327	327 <sup>†</sup>
18	414	419	422	417
20	520	535	529	520
22	626	633	*626	628
24	744	751	755	750
26	884	904	896	—
28	1021	1030	*1021	—
30	1170	1179	1184	—
32	1344	1369	1359	—
34	1512	1523	*1512	—
36	1692	1703	1709	—
38	1900	1930	1918	—
40	2099	2112	*2099	—
42	2310	2323	2330	—
44	2552	2587	2573	—
46	2782	2797	*2782	—
48	3024	3039	3047	—
50	3300	3340	3324	—

Old: the known best solutions in [12] as of August 2006

\* our solutions that were shown to be optimal

<sup>†</sup> already proved to be optimal in [7]

## 6 Conclusions

In this paper, we considered the constant distance traveling tournament problem (CDTTP), a simple variant of the traveling tournament problem (TTP). We proposed a lower bound of the optimal value of CDTTP. Using the lower bound, we showed that some existing and double round-robin tournaments we generated are optimal. We also proposed two algorithms to construct feasible tournaments. Our algorithms construct single round-robin tournaments, divide them into partial schedules with two or three slots, and then concatenate them to make double round-robin tournaments. The Modified Circle Method is a simple heuristic algorithm that runs in linear time in the size of output, and this algorithm produces asymptotic optimal solutions for CDTTP. The Minimum Break Method produced feasible solutions up to  $n \leq 50$ . In addition, for all instances of  $n \equiv 1 \pmod{3}$  teams, the Minimum Break Method generates an optimal solution if we can obtain single round-robin tournaments satisfying some conditions.

Our future work is to improve algorithms to construct feasible tournaments for CDTTP and to tackle other variants of TTP.

*Acknowledgement.* This work was partially supported by Grants-in-Aid for Scientific Research, by the Ministry of Education, Culture, Sports, Science and Technology of Japan.

## References

1. Anagnostopoulos, A., Michel, L., Van Hentenryck, P., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling* 9, 177–193 (2006)
2. de Werra, D.: Geography, games and graphs. *Discrete Applied Mathematics* 2, 327–337 (1980)
3. Easton, K., Nemhauser, G., Trick, M.: The traveling tournament problem: description and benchmarks. In: Walsh, T. (ed.) *CP 2001*. LNCS, vol. 2239, pp. 580–585. Springer, Heidelberg (2001)
4. ILOG: ILOG CPLEX 9.0 (2003)
5. Miyashiro, R., Iwasaki, H., Matsui, T.: Characterizing feasible pattern sets with a minimum number of breaks. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 78–99. Springer, Heidelberg (2003)
6. Miyashiro, R., Matsui, T.: A polynomial-time algorithm to find an equitable home-away assignment. *Operations Research Letters* 33, 235–241 (2005)
7. Rasmussen, R.V., Trick, M.A.: A Benders approach for the constrained minimum break problem. *European Journal of Operational Research* 177, 198–213 (2007)
8. Rasmussen, R.V., Trick, M.A.: The timetable constrained distance minimization problem. In: Beck, J.C., Smith, B.M. (eds.) *CPAIOR 2006*. LNCS, vol. 3990, pp. 167–181. Springer, Heidelberg (2006)
9. Rasmussen, R.V., Trick, M.A.: Round robin scheduling – a survey. Working Paper 2006/2, Department of Operations Research, University of Aarhus (2006)
10. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research* 179, 775–787 (2007)
11. Russell, R.A., Leung, J.M.Y.: Devising a cost effective schedule for a baseball league. *Operations Research* 42, 614–625 (1994)
12. Trick, M.: Challenge traveling tournament problem (2006), <http://mat.gsia.cmu.edu/TOURN/>
13. Urrutia, S., Ribeiro, C.C.: Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. *Discrete Applied Mathematics* 154, 1932–1938 (2006)
14. Van Hentenryck, P., Vergados, Y.: Traveling tournament scheduling: a systematic evaluation of simulated annealing. In: Beck, J.C., Smith, B.M. (eds.) *CPAIOR 2006*. LNCS, vol. 3990, pp. 228–243. Springer, Heidelberg (2006)

# Scheduling the Brazilian Soccer Tournament with Fairness and Broadcast Objectives

Celso C. Ribeiro<sup>1</sup> and Sebastián Urrutia<sup>2</sup>

<sup>1</sup> Department of Computer Science, Universidade Federal Fluminense,  
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil

`celso@inf.puc-rio.br`

<sup>2</sup> Department of Computer Science, Universidade Federal de Minas Gerais,  
Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-010, Brazil

`surrutia@dcc.ufmg.br`

**Abstract.** The Brazilian soccer tournament is organized every year by the Brazilian Soccer Confederation. Its major sponsor is TV Globo, the largest media group and television network in Brazil, which imposes constraints on the games to be broadcast. Scheduling the games of this tournament is a very constrained problem, with two objectives: breaks minimization (fairness) and the maximization of the revenues from TV broadcasting. We propose an integer programming decomposition strategy to solve this problem to optimality. Numerical results obtained for the 2005 and 2006 editions of the tournament are reported and compared.

## 1 Introduction

Soccer is the most widely practiced sport in Brazil. The yearly Brazilian soccer tournament is the most important sport event in the country. It is organized by the Brazilian Soccer Confederation (CBF). Its major sponsor is TV Globo, the largest media group and television network in Brazil, which imposes constraints on the games to be broadcast.

The most attractive games are those involving a subset of elite teams with more fans and, consequently, with larger broadcast shares. Games involving teams from São Paulo and Rio de Janeiro (the two largest cities in Brazil) are of special interest to TV Globo, due to larger revenues from advertising.

The competition lasts seven months and is structured as a compact mirrored double round robin (MDRR) tournament [3]. It is played by  $n$  teams, where  $n$  is an even number ( $n = 24$  in 2004,  $n = 22$  in 2005, and  $n = 20$  in 2006). There are  $2n - 2$  rounds and each team plays exactly once in each round. There are at most two rounds of games per week. Each team faces every other twice: once at home and the other away. If team  $a$  plays against team  $b$  at home (resp. away) in round  $k$ , with  $k < n$ , then team  $a$  plays against team  $b$  away (resp. at home) in round  $k + n - 1$ . See [3] for a recent survey on the sport scheduling literature.

The revenues and the attractiveness of the tournament strongly depend on the schedule of the games. The organizers and the sponsors search for a schedule optimizing two different objectives. CBF attempts to maximize fairness, by



minimizing the number of breaks during the tournament (breaks minimization objective). A break occurs whenever a team plays two consecutive home games or two consecutive away games, see e.g. [8]. TV Globo aims to maximize its revenues, by maximizing the number of relevant games it is able to broadcast (broadcast objective). The schedule must also satisfy a number of hard constraints.

We propose an integer programming solution approach for solving this scheduling problem, based on the generation of feasible home–away patterns. The detailed problem formulation is presented in Section 2. The solution strategy is described in Section 3. Numerical results obtained for real-life instances corresponding to the 2005 and 2006 editions of the tournament are reported and compared in Section 4. Concluding remarks are drawn in the last section.

## 2 Problem Statement

We consider both the 2005 and 2006 editions of the competition, with respectively  $n = 22$  and  $n = 20$  participating teams. Every team has a home city and some cities host more than one team. Some teams are considered and handled as elite teams, due to their number of fans, to the records of their previous participations in the tournament, and to the value of their players. There are weekend rounds and mid-week rounds.

São Paulo and Rio de Janeiro are the two largest cities in Brazil (with more fans and, consequently, generating larger revenues from advertising) and both of them have four elite teams. Games cannot be broadcast to the same city where they take place and only one game per round can be broadcast to each city. Consequently, TV Globo wants to broadcast to São Paulo (resp. Rio de Janeiro) games in which an elite team from São Paulo (resp. Rio de Janeiro) plays away against another elite team from another city. Such games will be referred to as TV games.

Belém is a city very far away from São Paulo and Rio de Janeiro. TV Globo is not willing to broadcast games taking place at Belém, due to the high logistical costs. As well as following the structure of a MDRR tournament, the schedule should also satisfy other hard constraints:

1. Every team playing at home (resp. away) in the first round plays away (resp. at home) in the last round.
2. Every team plays once at home and once away in the two first rounds and in the two last rounds.
3. After any number of rounds during the first half of the tournament, the difference between the number of home games and away games played by any team is either zero or one (i.e., the number of home and away games is always balanced in the first  $n - 1$  rounds).
4. Some pairs of teams with the same home city have complementary patterns (i.e., whenever one of them plays at home, the other plays away).
5. Flamengo and Fluminense (two elite teams from Rio de Janeiro that share the same stadium for their home games) have complementary patterns in the last four rounds.

6. Games between teams from the same city are not to be played in mid-week rounds or in the last six rounds (since they are among the most attractive games).
7. There is at least one elite team from Rio de Janeiro playing outside Rio de Janeiro and one elite team from São Paulo playing outside São Paulo in every round.
8. If in some round there is only one elite team from Rio de Janeiro (resp. São Paulo) playing outside Rio de Janeiro (resp. São Paulo), then this game should not be held in Belém.

The two objectives that must be optimized are the minimization of the number of breaks and the maximization of the number of rounds in which there is at least one TV game to be broadcast to São Paulo plus the number of rounds in which there is at least one TV game to be broadcast to Rio de Janeiro. Therefore, while the broadcast objective regards the number of relevant games that TV Globo is able to broadcast, the breaks minimization objective establishes the home-away equilibrium in the sequence of games played by each team.

The requirements described above in terms of constraints and objectives resulted from several meetings and discussions with the organizers of the tournament and, in particular, with officials of TV Globo.

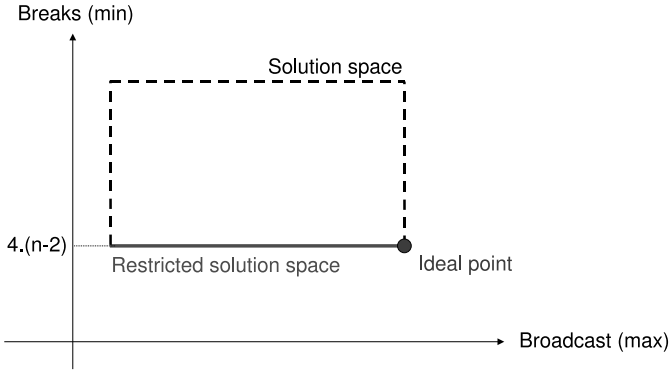
### 3 Solution Strategy

We propose the following approach to tackle this bi-objective problem. First, we add to the problem an extra constraint stating that the number of breaks is fixed at its minimum. Then, the broadcast objective is maximized with this additional constraint. If the maximum objective value of this restricted problem is equal to the unconstrained maximum, then this solution is what in multi-criteria optimization [4] is called an ideal point (all objectives are at their individual optimal values simultaneously). If the maximum solution value of the restricted problem is smaller than the unconstrained maximum, then the solution of the first is not an ideal point but still is a non-dominated solution (no other solution is better with respect to one of the objectives without being worse with respect to the other). Figure 1 illustrates this approach.

We assume that there exists at least one feasible solution with a minimum number of breaks: i.e., we assume that the restricted search space illustrated in Figure 1 is not empty. This fact was experimentally verified for all test instances and their variations.

#### 3.1 Bounds

Given a round robin schedule with  $r$  rounds, a home-away pattern (HAP) is a vector of  $r$  positions filled with ‘A’ and ‘H’. An ‘A’ (resp. ‘H’) in position  $s$  of a HAP indicates that every team associated with this HAP plays away (resp. at home) at round  $s$ . Since each team has to play against every other team, each



**Fig. 1.** Solution space and restricted solution space

team must be associated with a different HAP. Figure 2 shows a HAP set for six teams in a single round robin schedule.

We first show that constraints (1) and the mirrored structure impose that  $4(n - 2)$  is a lower bound to the number of breaks.

Since the tournament is mirrored, if the number of breaks in the first half of a HAP is even, then the total number of breaks is also even and equal to twice the number of breaks in the first half. On the other hand, if the number of breaks in the first half is odd, then the total number of breaks is also odd and equal to twice the number of breaks in the first half plus one (there is an extra break in the first round of the second half).

There are only two HAPs without breaks for single round robin schedules, one starting with a home game and the other starting with an away game. Therefore, the other  $n - 2$  teams must have at least one break in the first half of the schedule, yielding at least three breaks in the whole schedule. We notice that in double round robin schedules a team with an odd number of breaks plays its last game in the same playing condition of its first game (home-home or away-away). Therefore, to satisfy constraints (1), we shall consider schedules in which every team has an even number of breaks. In consequence, the  $n - 2$  teams having three breaks must have an extra break and the number of breaks cannot be smaller than  $4(n - 2)$ .

The broadcast objective can also be bounded. Since at most one TV game can be broadcast to São Paulo and another to Rio de Janeiro in every round, the broadcast objective cannot be greater than twice the number of rounds. Furthermore, the broadcast objective is also bounded by the number of existing TV games. The later is equal to the number of elite teams from São Paulo multiplied by the number of elite teams outside São Paulo, plus the number of elite teams from Rio the Janeiro multiplied by the number of elite teams outside Rio de Janeiro. The second bound is stronger (i.e., smaller) for the instances solved in this work. As an example, the first bound is equal to 84 for  $n = 22$ ,

---

Team 1:	A H A H A
Team 2:	H A H A H
Team 3:	A H H H A
Team 4:	H A A H H
Team 5:	H A A A H
Team 6:	A H H A A

---

**Fig. 2.** HAP set for a tournament with six teams

while the second is equal to 56 (four elite teams from Rio de Janeiro, four from São Paulo, and three from other cities).

### 3.2 Solution Algorithm

A straightforward integer programming formulation of the problem could not be solved by a commercial solver such as CPLEX after an entire day of computation.

Decomposition methods have been previously proposed for problems where the distances between the venues were not relevant. Nemhauser and Trick [6] proposed a three-phase scheme to exactly solve the problem of scheduling a basketball league. Feasible home-away patterns are created in the first phase. In the second phase, a different feasible HAP is assigned to each team (two different teams must have different HAPs in every feasible round robin schedule). Finally, in the last phase, the schedule is created respecting the previously determined HAP assignments.

Some recent papers dealt with scheduling problems in soccer tournaments. Della Croce and Oliveri [2] tackled the Italian soccer league. Bartsch et al. [1] worked on the schedule of the soccer leagues of Austria and Germany. Goossens and Spieksma [5] considered the scheduling of the Belgian soccer league. Noronha et al. [7] proposed a branch-and-cut algorithm to schedule the Chilean soccer tournament. The algorithms proposed in [1,2] follow a decomposition scheme similar to that of [6].

We propose an algorithm following an approach similar to the above described multi-phase decomposition scheme. Figure 3 illustrates this approach, whose four phases are described in the next sections.

### 3.3 Phase 1: HAP Generation

HAPs of mirrored schedules are divided into two symmetric halves. The second half is completely determined by the first. Therefore, we may determine which properties the first half of a HAP must obey so as that the entire HAP be feasible. As noticed in Section 3.1, if the number of breaks in the first half of a HAP is even, then the total number of breaks is also even and equal to twice the number of breaks in the first half. On the other hand, if the number of breaks in the first half is odd, then the total number of breaks is also odd and equal to twice the number of breaks in the first half plus one.

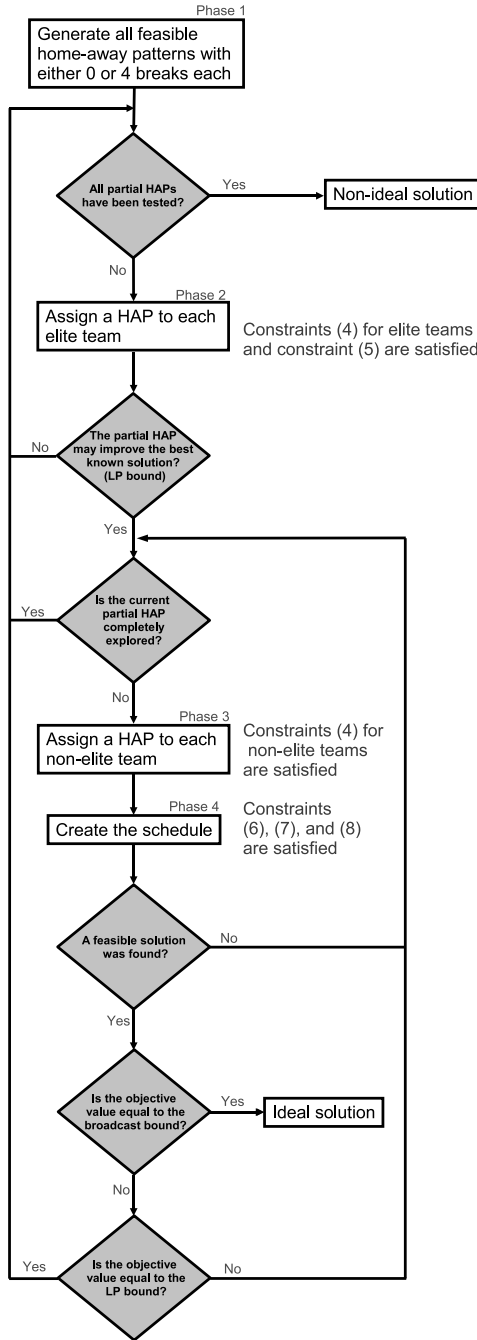


Fig. 3. Solution approach

HAPs satisfying constraints (1) are those with an even number of breaks. Therefore, we consider only HAPs with an even number of breaks in the first half. Since we are interested in schedules with a minimum number of breaks, we only consider HAPs with either zero (there are only two such HAPs) or two breaks in the first half.

HAPs satisfying constraints (2) are those without breaks in the second and last rounds. Since the schedule is mirrored, they must have no breaks in the last round of the first half (round  $n - 1$ ).

The difference between the number of home and away games in a HAP without breaks is equal to one after odd rounds and equal to zero after even rounds. If a HAP has a break in an even round, this difference increases to two. Therefore, HAPs satisfying constraints (3) are those without breaks in even rounds of the first half. Even (resp. odd) rounds of the second half are globally odd (resp. even) in mirrored schedules. In consequence, teams with breaks have at least one break in an even round and the difference between the number of home and away games will be necessarily greater than one after some round. For this reason, constraints (3) are limited to the first half of the schedule. If they were imposed to the whole schedule, the problem would be infeasible.

Consequently, feasible HAPs for the first half are those without breaks or with exactly two breaks in odd rounds (but not in the last, since constraints (2) forbid breaks in the last round of the second half). There are  $n/2 - 2$  rounds (all odd rounds but the first and the last) in which teams may have their two breaks, yielding a total of  $\binom{n/2-2}{2} = (n/2-2) \cdot (n/2-3)/2$  possible break configurations. Since there are two HAPs for every possible break configuration (one starting by a home game and the other by an away game), the number of feasible HAPs with two breaks is equal to  $(n/2 - 2) \cdot (n/2 - 3)$ . Considering the two HAPs without breaks, the total number of feasible HAPs is equal to  $(n/2 - 2) \cdot (n/2 - 3) + 2$ .

The number of feasible HAPs is equal to 58 for  $n = 20$  and to 74 for  $n = 22$ . This small number of feasible home-away patterns with at most two breaks each allows their complete enumeration in this phase.

### 3.4 Phase 2: Assignment of Partial HAPs to Elite Teams

In this phase, we use an explicit exhaustive enumeration to assign a HAP to each elite team satisfying constraints (4) and (5). The use of the two HAPs with no breaks is enforced, to keep the number of breaks at its minimum value  $4(n - 2)$ .

Constraint (4) is satisfied by assigning complementary HAPs to every pair of teams to which this constraint is imposed. Since half of the teams play at home and half away in every round of a feasible HAP assignment, in this phase and the next we first enumerate HAP assignments in which if one pattern is used, the complementary pattern is used as well. In this way, we improve the chance of a HAP assignment to be feasible.

Since Flamengo and Fluminense have their own complementary teams, they cannot have complementary patterns themselves. Therefore, to satisfy constraint (5) we assign patterns in such a way that (i) if Flamengo starts at home (resp. away), then Fluminense starts away (resp. at home), and (ii) either both

or none of them have a break in round  $n - 3$ . This ensures that Flamengo and Fluminense will have complementary patterns in the last four rounds, therefore satisfying constraint (5).

After assigning a HAP to each elite team, we build and solve a linear programming model considering the partial HAP assignments already made (recall that they satisfy constraints (1) to (5)), maximizing the broadcast objective subject to constraints (6) to (8). This linear programming model enforces that each elite team will play either at home or away in each round, depending on the HAP assigned to it.

The optimal value of the above linear program is an upper bound to the broadcast objective, associated to the current partial HAP assignment. This current partial assignment can be discarded if the bound it provides is smaller than the value of the broadcast objective for the current best known feasible solution. In this case, a new partial HAP assignment is enumerated and this phase is repeated for the new assignment. Otherwise, the algorithm proceeds to the third phase.

If all partial HAP assignments have been enumerated, the algorithm stops and returns the best feasible solution it obtained. The latter is non-dominated, but not an ideal solution.

### 3.5 Phase 3: Assignment of HAPs to Non-elite Teams

In this phase, once again we use an explicit exhaustive enumeration to assign one of the still available HAPs to each non-elite team, satisfying constraint (4) and completing the partial assignment constructed in the previous phase. Once the HAP assignment is complete, the algorithm proceeds to the last phase.

Whenever all possible alternatives to complete the partial assignment of HAPs to elite teams have been tested, the algorithm goes back to the second phase to enumerate a new partial assignment.

### 3.6 Phase 4: Schedule Creation

At this point, there is a HAP assigned to each team. In this phase, we build and solve an integer programming problem considering the current HAP assignments, maximizing the broadcast objective subject to constraints (6) to (8). This integer program defines the games to be played in each round, according to the home-away patterns assigned to each team.

In this straightforward problem formulation, we use binary variables  $x_{ijk}$  that are equal to one if and only if team  $i$  plays with team  $j$  at home in round  $k$ . Since at this phase we already know which teams play at home and which play away in each round, half of the variables are trivially equal to zero and can be eliminated. We also know that, due to the home-away pattern assigned to each team, the number of breaks is minimum and equal to  $4(n - 2)$ . Therefore, the number of variables is relatively small, since there is no need to use further variables to model the breaks. In consequence, this model can be quickly solved by a commercial solver.

**Table 1.** 2005 edition of the Brazilian tournament (22 teams)

	Official schedule	HAP-ILP schedule
Constraints (1)	yes	yes
Constraints (2)	yes	yes
Constraints (3)	no	yes
Constraints (4)	yes	yes
Constraint (5)	no	yes
Constraints (6)	no	yes
Constraints (7)	yes	yes
Constraints (8)	yes	yes
Breaks	156	80 (optimal)
Broadcast	43	56 (optimal)

We first assume that the above integer program is feasible. If its optimal value is equal to the broadcast bound, then the algorithm terminates with an ideal solution. Otherwise, if the optimal value of the integer program is equal to the linear programming bound obtained in the second phase, then the algorithm returns to the second phase to enumerate a new partial HAP assignment, because no better solution can be obtained with the current partial assignment.

If the integer program is infeasible or if its optimal value is smaller than the linear programming bound, then the algorithm returns to the third phase to enumerate another complete HAP assignment.

In any case, if the integer program is feasible and its optimal value is smaller than that of the current best known feasible solution, then the latter is updated.

## 4 Application to Real-Life Instances

The algorithm was applied to two instances corresponding to the 2005 and 2006 editions of the Brazilian tournament, with 22 and 20 teams respectively. There were 11 elite teams in each instance: four from São Paulo, four from Rio de Janeiro, and three from other cities.

CPLEX 9.0 was used as the linear and integer programming solver in the computational experiments. The algorithm was coded in C++, compiled with gcc and executed on a standard Pentium IV processor with 256 Mbytes of RAM memory.

The approach proposed in this work was able to compute optimal schedules providing ideal solutions (i.e., simultaneously optimizing both the broadcast and the breaks objectives) for the 2005 and 2006 editions of the tournament, always in less than ten minutes of execution time.

Tables 1 and 2 compare the schedules produced by the new algorithm (HAP-ILP) with those elaborated by the tournament organizers using ad hoc procedures based on their own expertise, acquired after many years creating the tournament schedule. The schedules are compared in terms of the satisfaction of the problem constraints and of the values of the two objective functions.



**Table 2.** 2006 edition of the Brazilian tournament (20 teams)

	Official schedule	HAP-ILP schedule
Constraints (1)	yes	yes
Constraints (2)	yes	yes
Constraints (3)	no	yes
Constraints (4)	yes	yes
Constraint (5)	no	yes
Constraints (6)	no	yes
Constraints (7)	yes	yes
Constraints (8)	yes	yes
Breaks	172	72 (optimal)
Broadcast	47	56 (optimal)

The solutions produced by our four-phase approach are clearly better than those produced by the current scheduler. Three types of constraints were not fully satisfied in the official schedules of the 2005 and 2006 editions of the tournament, while in both cases our algorithm provided optimal solutions satisfying all constraints. Regarding the 2005 edition, the ad hoc rules lead to schedules with 152 breaks and 43 TV games to be broadcast. The four-phase integer programming approach proposed in this work found a schedule with only 80 breaks (which is optimal), in which all 56 TV games could be broadcast (which is once again optimal). A similar comparison can be done for the 2006 edition of the tournament.

## 5 Conclusions

In this paper, we proposed a four-phase integer programming approach for scheduling the yearly Brazilian soccer tournament. This is a bi-criteria highly constrained mirrored round robin tournament, combining a fairness objective established by the organizers with an economical objective imposed by the TV sponsors.

The proposed algorithm was able to obtain optimal solutions maximizing both objectives and satisfying all constraints in a few minutes of computation time on a standard desktop computer. The schedules provided by the four-phase approach are clearly better than those currently used by the tournament organizers, which are obtained by simple ad hoc rules that are not even able to find feasible solutions satisfying all constraints.

A software system implementing a simple decision support system based on the proposed algorithm is able to generate a collection of same cost solutions to be evaluated and compared by the user. The use of this decision support system and the schedules created with the approach proposed in this work are currently under consideration by the tournament organizers.

## References

1. Bartsch, T., Drexler, A., Kröger, S.: Scheduling the professional soccer leagues of Austria and Germany. *Computers and Operations Research* 33, 1907–1937 (2006)
2. Della Croce, F., Oliveri, D.: Scheduling the Italian football league: An ILP-based approach. *Computers and Operations Research* 33, 1963–1974 (2006)
3. Easton, K., Nemhauser, G.L., Trick, M.: Sports scheduling. In: Leung, J.T. (ed.) *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pp. 52.1–52.19. CRC Press, Boca Raton, FL (2004)
4. Ehrgott, M.: *Multiple Criteria Optimization: Classification and Methodology*. Shaker, Aachen (1997)
5. Goossens, D., Spieksma, F.: Scheduling the Belgian soccer league. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 420–422 (August 2006)
6. Nemhauser, G.L., Trick, M.A.: Scheduling a major college basketball conference. *Operations Research* 46, 1–8 (1998)
7. Noronha, T.F., Ribeiro, C.C., Duran, G., Souyris, S., Weintraub, A.: A branch-and-cut algorithm for scheduling the highly constrained Chilean soccer tournament. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006*. LNCS, vol. 3867, pp. 174–186. Springer, Heidelberg (2007)
8. Urrutia, S., Ribeiro, C.C.: Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. *Discrete Applied Mathematics* 154, 1932–1938 (2006)

# Referee Assignment in Sports Leagues

Alexandre R. Duarte<sup>1</sup>, Celso C. Ribeiro<sup>2</sup>,  
Sebastián Urrutia<sup>3</sup>, and Edward H. Haeusler<sup>1</sup>

<sup>1</sup> Department of Computer Science, Catholic University of Rio de Janeiro,  
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil  
{[aduarte,hermann](mailto:aduarte@inf.puc-rio.br)}@inf.puc-rio.br

<sup>2</sup> Department of Computer Science, Universidade Federal Fluminense,  
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil  
[celso@inf.puc-rio.br](mailto:celso@inf.puc-rio.br)

<sup>3</sup> Department of Computer Science, Universidade Federal de Minas Gerais,  
Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-010, Brazil  
[surrutia@dcc.ufmg.br](mailto:surrutia@dcc.ufmg.br)

**Abstract.** Optimization in sports is a field of increasing interest. Combinatorial optimization techniques have been applied, for example, to game scheduling and playoff elimination. A common problem usually found in sports management is the assignment of referees to games already scheduled. There are a number of rules and objectives that should be taken into account when referees are assigned to games. We address a simplified version of a referee assignment problem common to many amateur leagues of sports such as soccer, baseball, and basketball. The problem is formulated by integer programming and its decision version is proved to be NP-complete. To tackle real-life large instances of the referee assignment problem, we propose a three-phase heuristic approach based on a constructive procedure, a repair heuristic to make solutions feasible, and a local search heuristic to improve feasible solutions. Numerical results on realistic instances are presented and discussed.

## 1 Introduction

Optimization in sports is a field of increasing interest. Some applications have been reviewed by Ribeiro and Urrutia [16]. Combinatorial optimization techniques have been applied, for example, to the traveling tournament problem [16,15,18], to playoff elimination [17], and to the scheduling of a college basketball conference [14]. Easton et al. [7] reviewed scheduling problems in sports.

A common problem usually found in amateur sports management is the assignment of referees to games already scheduled. Sport games are regulated by rules that depend on the sport and tournament. The *officiating crew* is a group of referees that is responsible to ensure that all rules are respected in a game. The number of referees compounding a crew may vary, depending on the sport, league, and tournament: soccer games usually require three referees, while basketball games require two. Each member of an officiating crew is said to occupy a *refereeing position* in a game. For example, in a regular soccer game, there

are one head umpire and two side judges, totaling three refereeing positions to be filled with referees. In some applications, managers make pre-assignments to satisfy some specific requirements. The referee assignment problem consists in assigning referees to the empty refereeing positions (not yet assigned) for all games of a league or tournament.

There are a number of rules and objectives that should be taken into account when referees are assigned to games. Games in higher divisions may require higher-skilled referees. Since referees may officiate several games during the day, travel feasibility and travel times between the facilities where the games take place have to be considered. Additionally, and especially in some amateur children leagues, some of the referees are players or their relatives. In this case, a natural constraint is that a referee cannot officiate a game in which he/she or a relative is scheduled to play.

Real-life versions of this problem appear in regional amateur leagues in the United States. Amateur leagues of several sports, such as baseball, basketball and soccer, have hundreds of games every weekend in different divisions. In a single league in California there might be up to 500 soccer games in a weekend, to be refereed by hundreds of certified referees. In the MOSA (Monmouth & Ocean Counties Soccer Association) league, New Jersey, boys and girls of ages 8 to 18 make up six divisions per age and gender group with six teams per division, totaling 396 games every Sunday.

Referee assignment problems in other contexts have been addressed in [8,9,19]. Dinitz and Stinson [4] considered a problem involving referee assignment to tournament schedules, connecting room squares and balanced tournament designs. We address a basic version of a referee assignment problem common to many amateur leagues of sports such as soccer, basketball, and baseball, among others. In the next section, we state the problem considered in this work. Section 3 presents an integer programming formulation to this referee assignment problem. The decision version of the problem is proved to be NP-complete in Section 4. The proposed solution strategy is described in Section 5. In Section 6, computational results illustrating the application of the proposed approach to solve real-size randomly generated instances are shown. Concluding remarks and further extensions of this work are reported in the last section.

## 2 Problem Statement

We consider the general problem, in which each game has a number of refereeing positions to be assigned to referees. The games are previously scheduled and the facilities and the time in which each game takes place are known beforehand. In our approach, referees are assigned to empty (i.e., not pre-assigned) refereeing positions, not to games. This allows us not only to handle referee assignment problems in sports requiring different numbers of referees, but also in tournaments where games of the same sport may need different numbers of referees due to the game division or importance. Games with pre-assigned referees to some refereeing positions can also be handled by this approach. Each refereeing position to be filled by a referee is called an empty refereeing slot.

Let  $S = \{1, \dots, n\}$  be the set of refereeing slots. Each refereeing slot  $j \in S$  has to be filled by a referee with a previously determined minimum skill level  $q_j$ . Let  $R = \{1, \dots, m\}$  be the set of referees, represented by their indices. Each referee  $i \in R$  has a certain skill level, denoted by  $p_i$ , defining the refereeing slots in which he/she can officiate. Referees may declare their unavailability to officiate at certain time slots. Furthermore, each referee  $i \in R$  establishes  $M_i$  as the maximum number of games he/she is able to officiate and  $T_i$  as the target number of games he/she is willing to officiate. Travels are not allowed, i.e. referees that officiate more than one game in the same day must be assigned to games that take place at the same facility. Moreover, referees that are also players have a hard facility assignment constraint: they must officiate at the same facility where they play.

The *Referee Assignment Problem* (RAP) consists in assigning referees to all refereeing slots associated to games scheduled in a given time interval (typically, a day or a weekend), minimizing the sum over all referees of the absolute value of the difference between the target and the actual number of games assigned to each referee and satisfying a set of hard constraints listed below:

- (a) all refereeing slots must be filled for all games;
- (b) referees cannot officiate more than one game in overlapping time slots;
- (c) referees cannot officiate games in time slots where they declared to be unavailable;
- (d) referees must meet the minimum skill level established for each refereeing slot;
- (e) referees cannot officiate more than a given maximum number of games; and
- (f) each referee can officiate in only one facility.

### 3 Integer Programming Model

The problem described in the previous section can be formulated by integer programming. We denote by  $d_i$  the absolute value of the difference between the target and the actual numbers of games assigned to referee  $i \in R$ . The following variables are used in the formulation:

$$x_{ij} = \begin{cases} 1, & \text{if referee } i \in R \text{ is assigned to slot } j \in S \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore,  $C(j) \subseteq S$  denotes the set of refereeing slots conflicting with slot  $j \in S$ , i.e. refereeing slots that take place at different facilities than or overlapping with  $j$ . Also,  $U(i) \subseteq S$  represents the set of refereeing slots to which referee  $i \in R$  cannot be assigned due to a lower skill level or to his/her unavailability. The RAP integer programming model can be formulated as

$$\text{minimize } \sum_{i=1}^m d_i \tag{1}$$

subject to:

$$d_i = |T_i - \sum_{j=1}^n x_{ij}| \quad \forall i = 1, \dots, m \quad (2)$$

$$\sum_{i=1}^m x_{ij} = 1 \quad \forall j = 1, \dots, n \quad (3)$$

$$\sum_{j=1}^n x_{ij} \leq M_i \quad \forall i = 1, \dots, m \quad (4)$$

$$x_{ij} + x_{ij'} \leq 1 \quad \forall i = 1, \dots, m, \quad \forall j = 1, \dots, n, \quad \forall j' \in C(j) \quad (5)$$

$$\sum_{j \in U(i)} x_{ij} = 0 \quad \forall i = 1, \dots, m \quad (6)$$

$$x_{ij} \in \{0, 1\} \quad \forall i = 1, \dots, m, \quad \forall j = 1, \dots, n. \quad (7)$$

The objective function (1) states that the sum over all referees of the slack between their target and actual numbers of scheduled games is minimized. Constraints (2) enforce that  $d_i$  is equal to the absolute value of the difference between the target and actual numbers of games assigned to referee  $i \in R$ . Constraints (3) ensure that every refereeing slot must be assigned to exactly one referee. Constraints (4) establish the upper bound to the number of refereeing slots that can be assigned to each referee. Constraints (5) ensure that refereeing slots with timetabling conflicts or taking place at different facilities cannot be assigned to the same referee. Constraints (6) prevent assignments that violate minimum skill level and unavailability restrictions (alternatively, all variables  $x_{ij}$  with  $j \in U(i)$  may simply be removed from the model). Constraints (7) establish the integrality of the decision variables. We notice that constraints (3) and (4) are those characterizing a generalized assignment problem [10].

## 4 NP-Completeness

We consider the following feasibility decision problem (DRAP):

**Problem:** REFEREE ASSIGNMENT

Input: Set  $S$  of refereeing slots, set  $R$  of referees, and the maximum number of games to be officiated by each referee.

Question: Is there an assignment of referees in  $R$  to refereeing slots in  $S$  satisfying constraints (a) to (f)?

**Theorem 1.** *DRAP is NP-complete.*

*Proof.* DRAP is clearly in NP, since the feasibility of any assignment can be checked in time polynomial in  $|R|$  and  $|S|$ . To prove its NP-completeness, we use a transformation from the problem of Partition into Bounded Independent Sets on interval graphs (PBIS). Given an undirected graph  $G = (V, E)$  and

integer numbers  $k$  and  $k'$ , this problem consists in deciding whether there exists a partition of  $V$  into  $k$  independent sets  $I_1, \dots, I_k$ , with  $|I_i| \leq k'$  for  $1 \leq i \leq k$ . PBIS is NP-complete even if  $G$  is an interval graph, see [2].

We build an instance of DRAP where the set  $S = \{1, \dots, |V|\}$  has exactly  $|V|$  refereeing slots, each of them associated with a different game. All games take place at the same date and facility. The minimum skill level associated to each refereeing slot  $j \in S$  is set as  $q_j = 1$ . Let  $R = \{1, \dots, k\}$  be the set of available referees and set  $M_i = k'$ ,  $p_i = 1$ , and  $U(i) = \emptyset$  for every  $i \in R$ . The linear time recognition algorithm of Corneil et al. [3] is used to build an interval representation of  $G$ . Each interval of the latter is mapped to one refereeing slot, whose starting and ending times coincide with the starting and ending points of the corresponding interval.

We now prove that given the interval graph  $G$  and the integer numbers  $k, k' \in N$ , there is a partition of  $V$  into independent sets  $I_1, \dots, I_k$  with  $|I_i| \leq k'$  for  $1 \leq i \leq k$  if, and only if, there is a feasible assignment of the referees in  $R$  to the set of refereeing slots  $S$  built as above, subject to constraints (a) to (f), with  $q_j = 1$  for all  $j \in S$ ,  $M_i = k'$  and  $p_i = 1$  for all  $i \in R$ .

First, suppose we have a partition of  $G = (V, E)$  into independent sets  $I_1, \dots, I_k$ , with  $|I_i| \leq k'$  for  $1 \leq i \leq k$ . The slots assigned to referee  $i \in R$  are exactly those corresponding to the vertices in  $I_i$ . This association guarantees that constraints (a) and (b) are satisfied. Constraints (c) and (f) are trivially satisfied, since  $U(i) = \emptyset$ , for all  $i \in R$ , and all games take place at the same facility. Since  $p_i = 1$ , for all  $i \in R$ , and  $q_j = 1$ , for all  $j \in S$ , constraint (d) is also trivially satisfied. Finally, constraint (e) is satisfied since  $|I_i| \leq k' = M_i$  for all  $i \in R$ .

We now consider a feasible solution to an instance of DRAP. We construct the interval graph  $G = (V, E)$  by assigning each refereeing slot  $j \in S$  to a vertex  $j \in V$ . There is an edge  $(j, j') \in E$  for each pair  $j$  and  $j'$  of overlapping refereeing slots. The partition of  $V$  into the independent sets  $I_1, \dots, I_k$  is such that vertex  $j$  belongs to the independent set  $I_i$  if referee  $i$  is assigned to refereeing slot  $j$ . As the number of slots assigned to each referee is bounded by  $k'$ , this partition into bounded independent sets is feasible.  $\square$

## 5 Solution Approach

We propose a three-phase heuristic approach to tackle real-life large instances of the referee assignment problem. The first phase consists in applying a greedy heuristic to find an initial solution, possibly violating some constraints. The second phase is a repair heuristic, which is applied whenever necessary to make the initial solution feasible. Finally, another heuristic is used in the third phase to improve the feasible solution. The algorithms used in the second and third phases are based on principles similar to the Iterated Local Search (ILS) meta-heuristic [11][12][13]. Algorithm 1 shows the general scheme of this approach.

The next section presents the constructive algorithm used to build initial solutions to the subsequent phases (repair and improvement). Section 5.2 describes

the ILS scheme which is the basis for both the repair and improvement phases. Details of the local search procedure used within the ILS scheme are given in Section 5.3. Sections 5.4 and 5.5 discuss some issues of the ILS scheme that are particular to each of the repair or improvement heuristics.

```

1 Algorithm
   RefereeAssignmentHeuristic(MaxIterations1, MaxIterations2)
2 Solution  $\leftarrow$  BuildGreedySolution();
3 if not isFeasible(Solution) then
4   | Solution  $\leftarrow$  RepairHeuristic(Solution, MaxIterations1);
5 end
6 if isFeasible(Solution) then
7   | Solution  $\leftarrow$ 
   |   ImprovementHeuristic(Solution, MaxIterations2);
8   | return Solution;
9 else
10  | return no feasible solution was found;
11 end

```

**Algorithm 1.** Referee assignment heuristic

## 5.1 Greedy Constructive Heuristic

The first phase of our approach attempts to build a feasible solution. Its main principle consists in assigning first the referees that are also players to the facilities where they have a game. Next, while there are unassigned refereeing slots and unassigned referees, the heuristic greedily selects a facility with unassigned refereeing slots, obtains an unassigned referee and assigns refereeing slots to this referee without violating any constraint. Finally, if any refereeing slot remains unassigned, the solution is completed with infeasible assignments.

The pseudo-code of this heuristic is presented in Algorithm 2. We denote by  $S^u$  the set of all unassigned refereeing slots, by  $R^{HF}$  the set of referees associated with a hard facility constraint, and by  $R^{NHF}$  the set of referees with no hard facility constraint, i.e.  $R = R^{HF} \cup R^{NHF}$ . These sets are initialized respectively in lines 2, 3, and 4. The loop in lines 5 to 15 is performed until all referees associated with hard facility constraints have been examined and assigned to as many refereeing slots as possible. Next, the loop in lines 16 to 27 attempts to fill the remaining unassigned refereeing slots with referees without hard facility constraints.

A greedy criterion is applied in line 18 to select a facility  $f$  with the strongest need for referees with a certain skill level  $\bar{p}$  computed in line 17. The computation of the greedy criterion is based on two measures: (a) an estimate of the minimum number of referees with skill level  $\bar{p}$  needed to officiate at facility  $f$  and (b) the number of unassigned refereeing slots in facility  $f$  with minimum skill level less than or equal to  $\bar{p}$ . Finally, if unassigned refereeing slots still remain at line 28, then the loop in lines 29 to 34 makes infeasible assignments to complete the solution.



```

1 Algorithm BuildGreedySolution()
2  $S^u \leftarrow \{j = 1, \dots, n : \sum_{i=1}^m x_{ij} = 0\}$ ;
3  $R^{HF} \leftarrow \{i = 1, \dots, m : \text{referee } i \text{ plays at least one game}\}$ ;
4  $R^{NHF} \leftarrow R - R^{HF}$ ;
5 while  $R^{HF} \neq \emptyset$  do
6   Randomly select a referee  $i \in R^{HF}$ ;
7    $R^{HF} \leftarrow R^{HF} - \{i\}$ ;
8   Let  $f$  be the facility where referee  $i$  plays a game;
9   forall  $j \in S^u$  : refereeing slot  $j$  takes place at facility  $f$ 
   do
10    if referee  $i$  can be assigned to refereeing slot  $j$  then
11       $x_{ij} \leftarrow 1$ ;
12       $S^u \leftarrow S^u - \{j\}$ ;
13    end
14  end
15 end
16 while  $S^u \neq \emptyset$  and  $R^{NHF} \neq \emptyset$  do
17    $\bar{p} \leftarrow \max_{i \in R^{NHF}} \{p_i\}$ ;
18   Let  $f$  be the facility with the strongest need for referees
   with skill level equal to  $\bar{p}$ ;
19   Randomly select a referee  $i \in R^{NHF}$  with  $p_i = \bar{p}$ ;
20    $R^{NHF} \leftarrow R^{NHF} - \{i\}$ ;
21   forall  $j \in S^u$  : refereeing slot  $j$  takes place at facility  $f$ 
   do
22    if referee  $i$  can be assigned to refereeing slot  $j$  then
23       $x_{ij} \leftarrow 1$ ;
24       $S^u \leftarrow S^u - \{j\}$ ;
25    end
26  end
27 end
28 if  $S^u \neq \emptyset$  then
29   forall  $s_j \in S^u$  do
30     Let  $f$  be the facility where  $s_j$  takes place;
31     Randomly select a referee  $i \in R$  officiating at facility
      $f$ ;
32      $x_{ij} \leftarrow 1$ ;
33      $S^u \leftarrow S^u - \{j\}$ ;
34   end
35 end
36 return Solution : referee  $i \in$ 
    $R$  is assigned to refereeing slot  $j \in S$  if and only if  $x_{ij} = 1$ ;
37

```

**Algorithm 2.** Greedy randomized constructive heuristic

```

1 Algorithm ILS_Scheme(Solution, MaxIterations)
2 foreach facility  $f$  do
3   | Solution  $\leftarrow$  LocalSearch( $f$ , Solution);
4 end
5 for  $i = 1, \dots, \text{MaxIterations}$  do
6   | NewSolution  $\leftarrow$  Perturbation(Solution);
7   | Let  $f_1$  and  $f_2$  be the facilities involved in the
   | perturbation;
8   | NewSolution  $\leftarrow$  LocalSearch( $f_1$ , NewSolution);
9   | NewSolution  $\leftarrow$  LocalSearch( $f_2$ , NewSolution);
10  | Solution  $\leftarrow$ 
   | AcceptanceCriterion(Solution, NewSolution);
11 end
12 return Solution;

```

**Algorithm 3.** ILS-based scheme

## 5.2 ILS-Based Scheme

Both the repair and the improvement heuristics use similar ILS schemes. They start by applying a first improving local search to the initial solution. Since the local search involves moves that change referee assignments for only one facility at a time, it should be applied to every facility.

Then, for a given number of iterations, a perturbation involving one pair of facilities is applied to the current solution. Each perturbation is followed by two applications of the local search procedure, once to each of the facilities of the pair involved in the perturbation. The solution obtained by local search is accepted if it satisfies a given acceptance criterion. This scheme is illustrated by the pseudo-code of Algorithm 3.

We describe next the local search procedure and its associated neighborhoods, followed by the repair and improvement heuristics.

## 5.3 Local Search and Neighborhoods

Solutions built by the constructive heuristic are not necessarily optimal or even feasible. A local search algorithm successively replaces the current solution by a better one in a neighborhood of the first, terminating at a local optimum. In a *first improving* strategy, the current solution is replaced by the first neighbor whose cost function value improves that of the current solution. We consider two neighborhoods for local search:

- swap moves: referees assigned to two refereeing slots are swapped (such moves do not change the number of games assigned to each referee) and
- replace moves: the referee assigned to a refereeing slot is replaced by another referee (such moves increase by one the number of games assigned to one referee and decrease by one the number of games assigned to the other).

As referees cannot be assigned to games at different facilities (hard constraint), only moves involving referees that officiate at the same facility (or do not officiate at all) are allowed (otherwise, and unless two referees were assigned to exactly one game each, a move involving referees that officiate at different facilities would imply at least one constraint violation). Such restricted neighborhoods considering only moves involving the same facility allow the acceleration of the local search.

The local search procedure performed within the ILS scheme is divided into two phases, both of them using a first improving strategy. In the first phase, only improving moves are accepted. The second phase also accepts moves leading to solutions at least as good as the current one, using a list of forbidden moves to prevent cycles. Each phase is separated in two parts: first, only swap moves are considered; next, only replace moves.

#### 5.4 Repair Heuristic

The repair heuristic follows the ILS scheme in Algorithm 3, based on local search and perturbations. It attempts to make feasible the initial solution obtained by the greedy randomized constructive heuristic. Constraint violations in the initial solution may concern time conflicts, referee unavailabilities, inadequate skill levels, or maximum numbers of games. The repair heuristic minimizes the number of constraint violations of an infeasible initial solution. A modified solution is feasible if and only if it has no constraint violations.

Solutions built by the constructive heuristic have the property that all referees officiate in at most one facility. Therefore, the local search considers only moves involving referees that officiate at the same facility (or do not officiate at all) and attempts to find a feasible solution by minimizing the number of constraint violations. Ties with respect to the number of constraint violations are broken in favor of the solution with the smaller objective function value (i.e., the absolute value of the difference between the target and the actual number of games assigned to each referee involved in a move).

The perturbation procedure within the repair heuristic changes the facility where one of the referees officiates, according to the following steps:

1. select a facility  $f$  with infeasible referee assignments;
2. select the highest minimum skill level  $\ell^*$  over all refereeing slots in facility  $f$  assigned to referees with at least one violation;
3. determine a referee  $r$  that officiates at another facility  $f'$  (or does not officiate at all) whose skill level is greater than or equal to  $\ell^*$ ;
4. randomly select referees other than  $r$  that officiate at facility  $f'$  and assign them to the refereeing slots currently assigned to  $r$ ;
5. assign referee  $r$  to any refereeing slot at facility  $f$  currently assigned to a referee with at least one violation.

The solution `NewSolution` obtained after a perturbation followed by local search is accepted by procedure *AcceptanceCriterion* if and only if it has fewer constraint violations or the same number of violations and a smaller objective function value than the current solution.

## 5.5 Improvement Heuristic

Once a feasible solution is known, the improvement heuristic is performed as an attempt to reduce the current value of the objective function, i.e. to minimize the sum over all referees of the absolute value of the difference between the target and the actual number of games assigned to each of them. The improvement heuristic is also based on the ILS scheme presented in Section 5.2.

The local search used in the improvement heuristic differs slightly from that used in the repair heuristic: swap moves are not performed (because they cannot improve the objective function) and only moves and perturbations that preserve feasibility are considered.

The perturbations applied to the current solution within the improvement heuristic select two referees that officiate at different facilities and swap all their assignments, according to the following steps:

1. Choose a possible perturbation: select two referees officiating at different facilities. If the swap of all their assignments does not preserve feasibility, then go to the final step. Otherwise, temporarily perform the swap of all assignments of the two selected referees.
2. Look ahead: for each of the two selected referees whose target number of games is greater than the new (after the swap) number of games he/she will officiate, check if there are other refereeing slots in which he/she could officiate at the new facility. This look-ahead procedure only checks refereeing slots that are currently assigned to referees officiating more games than their targets and only until the referee under investigation does not officiate more games than his/her target. Whenever possible, temporarily replace the previously assigned referee by the new referee involved in the perturbation.
3. Accept the perturbation: if the perturbation applied to the two referees (swap of all their assignments), followed by all possible replace moves in the destination facility for each referee, decreases the objective function value, then the perturbation is accepted and all temporary changes are made final. Otherwise, go to step 4.
4. Return: if all pairs of referees have already been considered, then perform the swap of all assignments of the pair of referees that increases the least the objective function, while preserving feasibility. Otherwise, return to the first step to select a new pair of referees.

Solution `NewSolution` obtained after a perturbation followed by local search is always accepted, because the heuristic chooses either an improving perturbation or the one that deteriorates the least the current solution.

## 6 Computational Results

Only very small instances with up to 40 games and 60 referees could be exactly solved by a commercial solver such as CPLEX 9.0, applied directly to the integer programming model presented in Section 3. In this section, we report computational results obtained on realistic, real-size randomly generated instances.

**Table 1.** Instance dimension combinations

Games	Referees	Facilities	Patterns
300	450, 525, 600	40, 50	$P_0, P_1$
400	600, 700, 800	55, 65	$P_0, P_1$
500	750, 875, 1000	65, 85	$P_0, P_1$

## 6.1 Test Problems

Since the RAP is a new problem, no benchmark instances are available. Test instances have been randomly generated, following patterns similar to those observed in real-life soccer instances. They have up to 500 games and up to 1000 referees, with different numbers of referees, different numbers of facilities, and different patterns of the target number of games each referee is willing to officiate.

Each game lasts for two hours and is scheduled to start at any hour from 8 AM to 7 PM. A facility and a starting time are randomly assigned to each game. There are three refereeing slots to be assigned to referees in each game: one of them requires a higher skill level (the head umpire), while the two other require less skilled referees (the two side judges).

The skill level  $p_i$  and the maximum number of games  $M_i$  for each referee  $i \in R$  are randomly generated in  $\{1, \dots, 8\}$  and  $\{2, \dots, 8\}$ , respectively. Two different patterns were used to generate the target number of games  $T_i$  for each referee  $i \in R$ . According to pattern  $P_0$ ,  $T_i$  is randomly selected from  $\{0, \dots, M_i\}$ . In the case of pattern  $P_1$ ,  $T_i$  is proportional to  $1/p_i$ : the higher the referee skill level is, the lower his/her target number of games is. This reasoning allows the creation of some challenging instances in which the more qualified a referee is, the lower is the number of games he/she wants to officiate.

Table 1 presents the parameter values used to generate the test problems. Five different instances were generated for each of the 36 parameter combinations, in a total of 180 test problems. All test problems are available from [5].

## 6.2 Numerical Results

The experimental results reported in this section were obtained on a 2.0 GHz Pentium IV processor with 512 Mbytes of RAM memory running Windows 2000<sup>TM</sup>. All codes were implemented in C. The maximum number of iterations performed by the repair and the improvement heuristics was set at 1000 and 200, respectively.

Tables 2-7 summarize the results obtained for some classes of test problems by the heuristic approach. We only report results for the hardest problems, in which the number of games (500) is large and the number of referees is limited (problems with 1,000 referees have been discarded). Initial solutions are computed by the greedy heuristic. Computation times (in seconds) and objective function values are average results over ten runs for each instance. For each phase of the

**Table 2.** 500 games, 750 referees, 65 facilities, and pattern  $P_0$

Instance	Construction			Repair			Improvement	
	Time (s)	Value	Feas.	Time (s)	Value	Feas.	Time (s)	Value
I <sub>1</sub>	0.02	1286.20	10	—	—	—	32.34	619.60
I <sub>2</sub>	0.02	1360.00	5	0.47	1338.00	10	31.81	623.40
I <sub>3</sub>	0.02	1269.00	2	0.60	1247.00	10	33.87	621.60
I <sub>4</sub>	0.03	—	—	1.14	1303.20	10	30.28	627.20
I <sub>5</sub>	0.03	1302.67	3	1.40	1259.14	10	33.73	654.00

**Table 3.** 500 games, 750 referees, 65 facilities, and pattern  $P_1$

Instance	Construction			Repair			Improvement	
	Time (s)	Value	Feas.	Time (s)	Value	Feas.	Time (s)	Value
I <sub>1</sub>	0.02	1752.75	8	0.66	1709.00	10	31.59	1022.60
I <sub>2</sub>	0.02	1669.57	7	0.02	1675.67	10	30.34	888.60
I <sub>3</sub>	0.02	—	—	5.91	1569.80	10	29.55	942.00
I <sub>4</sub>	0.03	1777.00	1	1.53	1725.00	10	31.81	1033.80
I <sub>5</sub>	0.02	1704.80	5	0.49	1704.80	10	28.17	952.00

heuristic (construction, repair, improvement), we present its computation time (in seconds) and the objective function value of the solutions found. For the construction and repair phases, we also report the number of runs where a feasible solution was found.

The constructive heuristic ran in less than 0.1 second for all instances and found feasible solutions for most of them. The repair heuristic found a feasible solution in less than 20 seconds in almost all cases when the constructive heuristic failed. This is due to the effectiveness of the constructive algorithm. Table 8 depicts some illustrative results on instances with 500 games, 750 referees, and 85 facilities to support this claim. For each instance, we show that the total times to build feasible solutions starting from randomly generated assignments are much larger than those observed when the initial solution is computed by the greedy constructive algorithm. We also observed that the repair phase failed to build feasible solutions from randomly generated initial solutions for some instances, even after 10,000 iterations, but always succeeded when starting from a solution built by the constructive heuristic. We stress the importance of quick procedures for finding initial solutions for hard combinatorial problems in sports, as already noticed by Ribeiro and Urrutia [18].

The improvement phase improved the objective function value of feasible initial solutions by up to 63%. Instances with more facilities or fewer referees were harder in terms of computation times and building feasible solutions.

In another experiment, we compared the results obtained by the heuristic with those found by CPLEX 9.0 when applied to formulation (1)–(7) for some small instances that could be exactly solved within reasonable computation time. The heuristic always received the same computation time that CPLEX took to find the optimal solution. Some numerical results are summarized in Table 9. For each

**Table 4.** 500 games, 750 referees, 85 facilities, and pattern  $P_0$

Instance	Construction			Repair			Improvement	
	Time (s)	Value	Feas.	Time (s)	Value	Feas.	Time (s)	Value
I <sub>1</sub>	0.03	—	—	11.27	1111.60	10	22.74	612.60
I <sub>2</sub>	0.03	—	—	6.69	1231.60	10	24.18	715.20
I <sub>3</sub>	0.03	—	—	11.33	1182.40	10	22.29	672.60
I <sub>4</sub>	0.03	—	—	4.61	1229.00	10	23.45	692.80
I <sub>5</sub>	0.03	—	—	3.39	1234.60	10	19.50	646.00

**Table 5.** 500 games, 750 referees, 85 facilities, and pattern  $P_1$

Instance	Construction			Repair			Improvement	
	Time (s)	Value	Feas.	Time (s)	Value	Feas.	Time (s)	Value
I <sub>1</sub>	0.03	—	—	2.75	1670.60	10	25.85	1043.80
I <sub>2</sub>	0.02	—	—	19.29	1649.50	8	26.15	1147.00
I <sub>3</sub>	0.03	—	—	14.77	1586.60	10	24.65	1107.60
I <sub>4</sub>	0.03	—	—	1.22	1602.80	10	25.59	1007.40
I <sub>5</sub>	0.03	—	—	2.69	1611.20	10	24.60	1002.80

instance, we first give its identification and the pattern used for its generation. The two next columns display the optimal solution value and the computation time in seconds taken by CPLEX (and, consequently, given to the heuristic). Next, the table shows the average and the best solution values found by the heuristic over ten runs. The last column gives the time taken to find the best solution in the corresponding run. These results show that the heuristic was able to find the optimal solution for three out of the five test instances considered in this table. Furthermore, the times taken by the heuristic are significantly smaller than those observed with CPLEX, even for the small instances that the latter was able to solve to optimality.

In the last computational experiment, we replaced the linear objective function by a quadratic penalization. Table 10 details the differences between the target and the actual numbers of games assigned to each referee in the solutions obtained with the linear and quadratic cost functions for instance I<sub>3</sub> with 500 games, 750 referees, 85 facilities, and pattern  $P_1$ . It shows that more balanced solutions can be obtained when the quadratic cost function is used, in which the occurrences of larger differences are replaced by those of smaller differences concentrated at only one unit. The computation times of the constructive, repair, and improvement heuristics were not affected by the change of the objective function. We observe that 76 extremely privileged referees (i.e., those officiating exactly their target number of games) in the solution obtained with the linear cost function lose their privileges in the solution obtained with the quadratic cost function. Also, 23 referees that were far from their targets are now very close to them (i.e., their differences are now equal to one). The new solution obtained with the quadratic cost function is certainly more fair than that associated with the linear costs.

**Table 6.** 500 games, 875 referees, 65 facilities, and pattern  $P_0$ 

Instance	Construction			Repair			Improvement	
	Time (s)	Value	Feas.	Time (s)	Value	Feas.	Time (s)	Value
I <sub>1</sub>	0.03	1582.80	10	—	—	—	45.07	574.40
I <sub>2</sub>	0.02	1627.40	10	—	—	—	42.92	609.20
I <sub>3</sub>	0.03	1535.40	10	—	—	—	44.62	558.20
I <sub>4</sub>	0.03	1655.60	10	—	—	—	43.28	576.60
I <sub>5</sub>	0.02	1626.00	10	—	—	—	43.31	619.20

**Table 7.** 500 games, 875 referees, 65 facilities, and pattern  $P_1$ 

Instance	Construction			Repair			Improvement	
	Time (s)	Value	Feas.	Time (s)	Value	Feas.	Time (s)	Value
I <sub>1</sub>	0.03	2195.20	10	—	—	—	39.64	1091.80
I <sub>2</sub>	0.03	2040.20	10	—	—	—	42.33	955.40
I <sub>3</sub>	0.02	2153.40	10	—	—	—	41.34	1032.20
I <sub>4</sub>	0.03	2173.40	10	—	—	—	42.54	1069.00
I <sub>5</sub>	0.03	2137.60	10	—	—	—	39.73	1035.00

**Table 8.** Greedy versus randomly generated initial solutions

Instance	Pattern	Greedy			Random	
		Const. (s)	Repair (s)	Feas.	Repair (s)	Feas.
I <sub>1</sub>	$P_0$	0.03	11.27	10	79.8	9
I <sub>2</sub>	$P_0$	0.03	6.69	10	80.8	10
I <sub>3</sub>	$P_0$	0.03	11.33	10	86.2	8
I <sub>4</sub>	$P_0$	0.03	4.61	10	30.6	10
I <sub>5</sub>	$P_0$	0.03	3.39	10	29.1	10
I <sub>1</sub>	$P_1$	0.03	2.75	10	33.5	10
I <sub>2</sub>	$P_1$	0.02	19.29	10	134.6	2
I <sub>3</sub>	$P_1$	0.03	14.77	10	135.1	8
I <sub>4</sub>	$P_1$	0.03	1.22	10	38.0	10
I <sub>5</sub>	$P_1$	0.03	2.69	10	32.9	10

**Table 9.** 33 games, 57 referees, 5 facilities, patterns  $P_0$  and  $P_1$ 

Instance	Pattern	CPLEX		Heuristic		
		Optimum	Time (s)	Average	Best	Time (s)
I <sub>2</sub>	$P_0$	43	164.00	47.00	44	18.99
I <sub>3</sub>	$P_0$	18	200.00	20.80	18	3.05
I <sub>5</sub>	$P_0$	44	137.00	45.20	44	11.45
I <sub>2</sub>	$P_1$	65	128.00	67.20	65	15.32
I <sub>5</sub>	$P_1$	72	47.00	82.40	75	8.83



**Table 10.** Linear versus quadratic objective functions

Difference from target	Number of referees	
	Linear penalties	Quadratic penalties
0	255	179
1	182	281
2	156	149
3	67	66
4	50	43
5	23	18
6	13	10
7	3	3

## 7 Concluding Remarks

We introduced in this paper the referee assignment problem, a new optimization problem in sports. The problem was formulated as an integer model and the NP-completeness of its decision version was proved.

A three-phase heuristic was proposed and implemented. Computational results on realistic instances showed the effectiveness of the greedy constructive heuristic combined with the repair heuristic to build feasible solutions. The improvement procedure used in the third phase was able to substantially improve solution quality. We also illustrated the importance of a quick construction procedure to build initial solutions.

We also compared the solutions obtained by the heuristic with those produced by CPLEX for some small instances that could be solved to optimality in reasonable computation times. The heuristic not only was able to find the optimal solutions for several instances, but also the computation times to find the best solution were significantly smaller than those observed with CPLEX.

Finally, we investigated and compared the behavior of an alternative quadratic objective function, which was able to find more fair solutions than the formulation with a linear cost function.

We are currently working on some extensions addressing further constraints of real-life applications, such as the existence of hard and soft links between some referees. In these situations, some referees may want to work with the same referees as partners in every game they officiate. This is the case when they are more confident to officiate together, but also when they want to travel in car pools or to officiate with relatives. Decision makers may also want referee assignments matching preferences regarding the facilities, divisions, and time slots where the referees officiate.

Another extension occurs when referees are able to officiate games in different facilities. In this case, travel times between facilities should also be considered for feasibility matters. They can also be incorporated to the objective function, so as that the minimization of the total traveling time turns out to be another objective. The minimization of the waiting times between consecutive games assigned to the same referee is also relevant.

The referee assignment problem has clearly the flavor of a multi-criteria optimization application. We are also investigating the use of multi-criteria methods coupled with a decision support system for its solution in practice.

## References

1. Anagnostopoulos, A., Michel, L., Van Hentenryck, P., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling* 9, 177–193 (2006)
2. Bodlaender, H.L., Jansen, K.: Restrictions of graph partition problems – Part I. *Theoretical Computer Science* 148, 93–109 (1995)
3. Corniel, D.G., Olariu, S., Stewart, L.: The ultimate interval graph recognition algorithm? In: *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 175–180. SIAM, Philadelphia (1998)
4. Dinitz, J.H., Stinson, D.R.: On assigning referees to tournament schedules. *Bulletin of the Institute of Combinatorics and its Applications* 44, 22–28 (2005)
5. Duarte, A.R.: Challenge referee assignment problem instances (last visited on March 23, 2007), Online document at <http://www.esportemax.org/rapopt>
6. Easton, K., Nemhauser, G.L., Trick, M.: The traveling tournament problem: Description and benchmarks. In: Walsh, T. (ed.) *CP 2001*. LNCS, vol. 2239, pp. 580–584. Springer, Heidelberg (2001)
7. Easton, K., Nemhauser, G.L., Trick, M.: Sports scheduling. In: Leung, J.T. (ed.) *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pp. 52.1–52.19. CRC Press, Boca Raton, FL (2004)
8. Evans, J.R.: A microcomputer-based decision support system for scheduling umpires in the American baseball league. *Interfaces* 18, 42–51 (1988)
9. Evans, J.R., Hebert, J.E., Deckro, R.F.: Play ball – The scheduling of sports officials. *Perspectives in Computing* 4, 18–29 (1984)
10. Fisher, M.L., Jaikumar, R., Van Wassenhove, L.N.: A multiplier adjustment method for the generalized assignment problem. *Management Science* 32, 1095–1103 (1986)
11. Lourenço, H.P., Martin, O., Stützle, T.: Iterated Local Search. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 321–353. Kluwer, Dordrecht (2003)
12. Martin, O., Otto, S.W.: Combining simulated annealing with local search heuristics. *Annals of Operations Research* 63, 57–75 (1996)
13. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the traveling salesman problem. *Complex Systems* 5, 299–326 (1991)
14. Nemhauser, G.L., Trick, M.A.: Scheduling a major college basketball conference. *Operations Research* 46, 1–8 (1997)
15. Rasmussen, R.V., Trick, M.A.: Round robin scheduling – A survey. Technical Report, Department of Operations Research, University of Aarhus (2006)
16. Ribeiro, C.C., Urrutia, S.: OR on the ball: Applications in sports scheduling and management. *OR/MS Today* 31, 50–54 (2004)
17. Ribeiro, C.C., Urrutia, S.: An application of integer programming to playoff elimination in football championships. *International Transactions in Operational Research* 12, 375–386 (2005)
18. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research* 179, 775–787 (2007)
19. Wright, M.B.: Scheduling English cricket umpires. *Journal of the Operational Research Society* 42, 447–452 (1991)

# A Branch-and-Cut Algorithm for Scheduling the Highly-Constrained Chilean Soccer Tournament

Thiago F. Noronha<sup>1</sup>, Celso C. Ribeiro<sup>2</sup>, Guillermo Duran<sup>3</sup>, Sebastian Souyris<sup>3</sup>,  
and Andres Weintraub<sup>3</sup>

<sup>1</sup> Department of Computer Science, Catholic University of Rio de Janeiro,  
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil  
`tfn@inf.puc-rio.br`

<sup>2</sup> Department of Computer Science, Universidade Federal Fluminense,  
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil  
`celso@inf.puc-rio.br`

<sup>3</sup> Department of Industrial Engineering, University of Chile,  
Republica 701, Santiago, Chile  
`{gduran,ssouyris,aweintra}@dii.uchile.cl`

**Abstract.** The qualifying phase of the Chilean soccer championship follows the structure of a compact single round robin tournament. Good schedules are of major importance for the success of the tournament, making them more balanced, profitable, and attractive. The schedules were prepared by ad hoc procedures until 2004, when a rough integer programming strategy was proposed. In this work, we improve the original integer programming formulation. We derive valid inequalities for improving the linear relaxation bound and we propose a new branch-and-cut strategy for the problem. Computational results on a real-life instance illustrate the effectiveness of the approach and the improvement in solution quality.

## 1 Introduction

There are 20 teams in the first division of the Chilean national soccer championship, organized by the National Association of Professional Soccer (ANFP). It is organized in two phases: qualifying and playoffs. The qualifying phase follows the structure of a compact single round robin tournament, in which each team plays against every other exactly once and all teams play exactly one game in every round. The teams are evenly distributed over four groups with five teams each. The groups are formed according to the performance of each team in the last tournament. Each of the first four teams is placed in one of the four groups. The teams from the 5th to the 8th places are randomly distributed in different groups. The same happens with the teams from the 9th to the 12th places. This procedure is repeated until all teams are assigned to a group. At the end of the qualifying phase, the teams that end up in the two first positions of each group qualify for the playoffs. The qualified teams play four quarter-final matches, whose winners play the two semi-final matches. The winners of the semi-final

matches play the final match. Playoff matches consist of two games each, each of them played at the home of one of the opponents.

The schedules of the Chilean soccer championship were prepared by ad hoc procedures until 2004. As for most European and South American soccer championships, the games were randomly assigned to slots in a predefined round sheet. Then, team representatives voted whether the schedule should be accepted or not. If the proposed schedule was not accepted, then the representatives proposed modifications and the voting process was repeated until a schedule was accepted by more than 50% of the representatives. There were several drawbacks with these schedules that made them less attractive for fans and less profitable for teams: (a) classical games at inconvenient rounds, (b) weak teams playing away all their games against strong teams, (c) teams playing too many consecutive home games or too many consecutive away games, and (d) no games between traditional teams and teams from tourist cities during summer rounds, when many people are visiting the tourist regions. Duran et al. [6] tackled the problem of scheduling the Chilean soccer championship by integer programming in 2005, handling the above issues. The model was solved by a standard branch-and-cut procedure of the CPLEX solver. However, this procedure would take up to two hours of computation time to find a feasible solution. The procedure would be interrupted at this time, the set of possible home-away patterns fixed, and the resulting simplified model solved to optimality using a limited set of decision variables. In consequence, the model becomes easy and solvable in a few seconds. Although the resulting schedules were better than those obtained by the ad hoc procedures, the duality gaps could be very large and solutions lacked of quality.

Good schedules have a major importance in the success of sports tournaments, making them more balanced, profitable, and attractive. Many authors tackled the problem of tournament scheduling optimization in different leagues and sports. Bean and Birge [2] focused on the scheduling problem for the National Basketball Association, in which the most limiting constraints concerned rest days and stadium availability. Costa [4] considered the scheduling of the National Hockey League, for which one of the objectives consisted in the minimization of the total distance traveled by all teams. Henz [8] used constraint programming to improve the processing times of the enumerative approach proposed in [10] to compute schedules for a college basketball conference. These results were later improved by Zhang [15], once again using constraint programming. We refer to Henz [9] for recent advances in constraint programming for scheduling problems in sports, as well as to Trick [13,14] for the combination of integer and constraint programming. Bartsch et al. [1] developed a branch-and-bound procedure for scheduling the professional soccer leagues of Austria and Germany. Goossens and Spieksma [7] proposed an integer programming formulation for scheduling the Belgian soccer league, whose objective function consisted in the minimization of the violations of soft constraints. Ribeiro and Urrutia [12] solved the problem of scheduling the Brazilian soccer tournament by an approach combining backtracking and integer programming, which found

optimal solutions very quickly. Croce and Oliveri [5] used a three-phase strategy based on integer programming for scheduling the Italian major soccer league, involving round robin and television constraints and minimizing the number of violations of home-away pattern constraints.

In this work, we tackle the problem of scheduling the highly-constrained Chilean soccer tournament. The original integer programming formulation of Duran et al. [6] is improved and valid inequalities are derived to strengthen the linear relaxation bound. We also developed a new branch-and-cut strategy that finds much better results than the previous approach.

The integer programming formulation is presented in Section 2. The solution approach and the branching strategy are described in Section 3. Computational results are reported in Section 4. Concluding remarks are made in the last section.

## 2 Problem Formulation

In this section, the problem of scheduling the Chilean soccer tournament is stated. We present the constants, variables, and constraints of the mathematical formulation, as well as its objective function. We first recall some widely used terminology in sports scheduling. A single round robin tournament is one in which each team plays against every other exactly once. A round robin tournament is compact if every team plays exactly once in each round. A home-away pattern (HAP) is a sequence of home and away games for a given team. A break is a subsequence of two consecutive home games or two away games.

The following subsets of teams are defined:

- *POP*: popular teams, which are those with more fans;
- *STR*: strong teams, better qualified in the last tournaments;
- *TRD*: traditional teams (Universidad Católica, Colo-Colo, and Universidad de Chile);
- *STG*: teams whose home city is Santiago; and
- *TUR*: teams from tourist cities, visited in summer and holidays.

Some constraints involve games and relationships between specific pairs of teams:

- *CMP*: pairs of teams with complementary HAPs (whenever one of them plays at home the other plays away, and vice versa);
- *EXC*: pairs of excluding teams (whenever a third team plays against one of them at home, then it should play away against the other, and vice versa); and
- *GRP*: pairs of teams in a same group.

Since some constraints involve some specific rounds, we also define

- *SUM*: summer rounds; and
- *WED*: Wednesday rounds.

Chile is geographically divided into thirteen regions (numbered from 1 to 13) and three zones (North, South, and Central):

- $TRG = \{4, 5\}$ : tourist regions;
- $ZNS = \{\text{North, South, Central}\}$ : zones;
- $FRG(r)$ : teams whose home cities are in region  $r$ ; and
- $FZN(z)$ : teams whose home cities are in zone  $z$ .

We first define the following decision variables:

$$x_{ijk} = \begin{cases} 1, & \text{if team } i \text{ plays at home against team } j \text{ in round } k, \\ 0, & \text{otherwise;} \end{cases}$$

$$y_{ik} = \begin{cases} 1, & \text{if team } i \text{ has an away break in round } k + 1, \\ 0, & \text{otherwise.} \end{cases}$$

Round robin constraints are those defining a timetable in which (i) each team plays against every other team exactly once (constraint (1)) and (ii) every team plays exactly once in each round (constraint (2)):

$$\sum_{k=1}^{19} (x_{ijk} + x_{jik}) = 1 \quad \forall i, j = 1, \dots, 20, \quad \text{with } i < j \quad (1)$$

$$\sum_{\substack{j=1 \\ j < i}}^{20} (x_{ijk} + x_{jik}) = 1 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19. \quad (2)$$

HAP constraints restrict the home–away patterns, imposing a fair balance between home and away matches: (i) each team must play at least nine (and at most ten) games at home and the others away (constraint (3)), (ii) a team may never have two consecutive breaks (constraints (4) and (5)), (iii) a team may play at most three games at home in any five consecutive rounds (constraint (6)), (iv) there may be no breaks in rounds 2, 17, and 19 (beginning and end of the tournament, constraint (7)):

$$9 \leq \sum_{\substack{j=1 \\ j \neq i}}^{20} \sum_{k=1}^{19} x_{ijk} \leq 10 \quad \forall i = 1, \dots, 20 \quad (3)$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{20} (x_{ij(k-1)} + x_{ijk} + x_{ij(k+1)}) \leq 2 \quad \forall i = 1, \dots, 20, \quad \forall k = 2, \dots, 18 \quad (4)$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{20} (x_{ji(k-1)} + x_{jik} + x_{ji(k+1)}) \leq 2 \quad \forall i = 1, \dots, 20, \quad \forall k = 2, \dots, 18 \quad (5)$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{20} (x_{ij(k-2)} + x_{ij(k-1)} + x_{ijk} + x_{ij(k+1)} + x_{ij(k+2)}) \leq 3 \quad \forall i = 1, \dots, 20, \quad \forall k = 3, \dots, 17 \quad (6)$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{20} (x_{ij(k-1)} + x_{ijk}) = 1 \quad \forall i = 1, \dots, 20, \quad \forall k = 2, 17, 19. \quad (7)$$

Consecutive away games are very inconvenient and should be avoided. Constraints (8) and (9) impose that every team should have at most one away break:

$$\sum_{\substack{j=1 \\ j \neq i}}^{20} (x_{jik} + x_{ji(k+1)}) \leq 1 + y_{ik} \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 18 \quad (8)$$

$$\sum_{k=1}^{18} y_{ik} \leq 1 \quad \forall i = 1, \dots, 20. \quad (9)$$

The last two HAP constraints guarantee that (i) some pairs of teams must have complementary HAPs (constraint (10)), for security reasons to avoid more than one game in the same city at the same time, to ensure that there will ever be one game in this city or because they share the same stadium, and (ii) there may be at most four teams from Santiago playing at home in any round (constraint (11)):

$$\sum_{\substack{h=1 \\ h \neq i, h \neq j}}^{20} (x_{ihk} + x_{jhk}) = \sum_{\substack{h=1 \\ h \neq i, h \neq j}}^{20} (x_{hik} + x_{hjk}) \quad \forall (i, j) \in CMP, \quad \forall k = 1, \dots, 19 \quad (10)$$

$$\sum_{i \in STG} \sum_{\substack{j=1 \\ j \neq i}}^{20} x_{ijk} \leq 4 \quad \forall k = 1, \dots, 19. \quad (11)$$

Team constraints restrict the rounds in which games between special pairs of teams can be played: (i) each team should play at least one game between two consecutive games against popular teams (constraint (12)), (ii) each team may have at most two consecutive games against strong teams (constraint (13)), (iii) each traditional team plays exactly one classical game (i.e., a game against another traditional team) at home (constraint (14)), and (iv) classical games must be played between rounds 8 and 17 (constraint (15)):

$$\sum_{j \in POP \setminus \{i\}} (x_{ijk} + x_{jik} + x_{ij(k+1)} + x_{ji(k+1)}) \leq 1 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 18 \quad (12)$$

$$\sum_{j \in STR \setminus \{i\}} (x_{ijk} + x_{jik} + x_{ij(k+1)} + x_{ji(k+1)} + x_{ij(k+2)} + x_{ji(k+2)}) \leq 2 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 17 \quad (13)$$

$$\sum_{k=1}^{19} (x_{hik} + x_{jik}) = \sum_{k=1}^{19} (x_{hjk} + x_{ijk}) \quad \forall i, j, h \in TRD \quad (14)$$

$$\sum_{i \in TRD} \sum_{\substack{j \in TRD \\ j \neq i}} \left( \sum_{k=1}^7 x_{ijk} + \sum_{k=18}^{19} x_{ijk} \right) = 0. \quad (15)$$

The strong teams are grouped into pairs to balance the hardness of home and away games. Whenever a team plays at home against one of the teams of a pair of excluding teams, then it will play away against the other (and vice-versa), as stated by constraints (16):

$$\sum_{k=1}^{19} (x_{hik} + x_{hjk}) = 1 \quad \forall (i, j) \in EXC, \quad \forall h = 1, \dots, 20, h \neq i, h \neq j. \quad (16)$$

Geographic constraints tackle Chile’s particular geography of a very long and narrow country: a team from the Central zone cannot play away in the same week against a team from the South and another from the North, and vice versa (constraints (17) and (18)). Whenever a team from the Central zone plays against a team from the North (resp. South) on a Wednesday, then it cannot play against a team from the South (resp. North) in the previous or forthcoming weekend. Furthermore, we point out that the first and last rounds are always scheduled on weekends:

$$\sum_{j \in FZN(\text{South})} (x_{ji(k-1)} + x_{ji(k+1)}) + \sum_{j \in FZN(\text{North})} 2 \cdot x_{jik} \leq 2 \quad \forall i \in FZN(\text{Central}), \quad \forall k \in WED \quad (17)$$

$$\sum_{j \in FZN(\text{North})} (x_{ji(k-1)} + x_{ji(k+1)}) + \sum_{j \in FZN(\text{South})} 2 \cdot x_{jik} \leq 2 \quad \forall i \in FZN(\text{Central}), \quad \forall k \in WED. \quad (18)$$

There are also constraints on tourist teams and regions: (i) each tourist team should play at least once at home against a traditional team during the summer rounds (constraint (19)) and (ii) each traditional team should not play twice in the same week in the same tourist region (constraint (20)):

$$\sum_{k \in SUM} \sum_{j \in TRD \setminus \{i\}} x_{ijk} \geq 1 \quad \forall i \in TUR \quad (19)$$



$$\sum_{i \in FRG(r) \setminus \{j\}} (x_{ij(k-1)} + 2 \cdot x_{ijk} + x_{ij(k+1)}) \leq 2$$

$$\forall j \in TRD, \quad \forall r \in TRG, \quad \forall k \in WED. \tag{20}$$

Since only the teams in the two first positions of each group qualify for the playoffs, games between teams in the same group are more attractive. Therefore, these games should as much as possible take place at the end of the tournament. The objective function (21) consists in maximizing the number of games between teams in the same group in the last rounds of the tournament:

$$\text{maximize} \quad \sum_{(i,j) \in GRP} \sum_{k=1}^{19} k \cdot x_{ijk}. \tag{21}$$

Duran et al. [6] attempted to apply a standard CPLEX branch-and-cut algorithm directly to the above formulation. However, the lower bounds provided by its linear relaxation were poor because of the flow spread among the originally binary  $x$  variables, which end up assuming fractional values. CPLEX heuristics were not able to find primal solutions. The computation times were very high, because the formulation is degenerate for the simplex method and can only be solved by perturbation techniques.

### 3 Solution Approach

The original integer programming formulation described in the previous section can be significantly improved. Valid inequalities are derived in Section 3.1 to improve the lower bounds and a new branch-and-cut strategy is proposed in Section 3.2 to speed up convergence.

#### 3.1 Improved Formulation

We first introduce the following additional binary variables:

$$z_{ik} = \begin{cases} 1, & \text{if team } i \text{ plays at home in round } k; \\ 0, & \text{otherwise.} \end{cases}$$

They can be related to the other variables by constraints (22) and (23):

$$z_{ik} = \sum_{\substack{j=1 \\ j \neq i}}^{20} x_{ijk} \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19 \tag{22}$$

$$z_{ik} + z_{i(k+1)} \geq 1 - y_{ik} \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 18. \tag{23}$$

All HAP constraints may be rewritten in terms of the new variables by substituting  $z_{ik} = \sum_{j \neq i, j=1}^{20} x_{ijk}$ : constraint (3) is rewritten as (24), constraints (4) and (5) as (25), constraint (6) as (26), constraint (7) as (27), constraint (8) as (28), constraint (10) as (28), and constraint (11) as (29):

$$9 \leq \sum_{k=1}^{19} z_{ik} \leq 10 \quad \forall i = 1, \dots, 19 \quad (24)$$

$$1 \leq z_{i(k-1)} + z_{ik} + z_{i(k+1)} \leq 2 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19 \quad (25)$$

$$z_{i(k-2)} + z_{i(k-1)} + z_{ik} + z_{i(k+1)} + z_{i(k+2)} \leq 3 \\ \forall i = 1, \dots, 20, \quad \forall k = 2, \dots, 19 \quad (26)$$

$$z_{i(k-1)} + z_{ik} = 1 \quad \forall i = 1, \dots, 20, \quad \forall k = 2, 17, 19 \quad (27)$$

$$z_{ik} + z_{jk} = 1 \quad \forall (i, j) \in CMP, \quad \forall k = 1, \dots, 19 \quad (28)$$

$$\sum_{i \in STG} z_{ik} \leq 4 \quad \forall k = 1, \dots, 19. \quad (29)$$

To reduce the number of fractional variables, we use an approach similar to that proposed by Trick [13], with the exception that the rounds in which the games will be played are not fixed. Additional variables are added as follows:

$$s_i = \begin{cases} 1, & \text{if team } i \text{ plays at home in the first round,} \\ 0, & \text{otherwise;} \end{cases}$$

$$h_{ik} = \begin{cases} 1, & \text{if team } i \text{ plays away in round } k \text{ and at home in round } k + 1, \\ 0, & \text{otherwise;} \end{cases}$$

$$w_{ik} = \begin{cases} 1, & \text{if team } i \text{ plays at home in round } k \text{ and away in round } k + 1, \\ 0, & \text{otherwise.} \end{cases}$$

Variables  $z$ ,  $s$ ,  $h$ , and  $w$  are related by Equations (30) and (31) in the same way as in [13]:

$$z_{ik} = s_i + \sum_{t=1}^{k-1} (h_{it} - w_{it}) \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19 \quad (30)$$

$$h_{ik} + w_{ik} \leq 1 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19. \quad (31)$$

Now, the following valid inequalities can be added:

$$z_{ik} + h_{ik} \leq 1 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19 \quad (32)$$

$$z_{ik} - w_{ik} \geq 0 \quad \forall i = 1, \dots, 20, \quad \forall k = 1, \dots, 19. \quad (33)$$

Constraints (32) and (33) do not improve the linear relaxation bound, but empirical observations showed that they diminish the number of non-integral  $z$  variables in the linear relaxation, speeding up the integer programming algorithm. The new formulation ((1),(2),(9),(12)–(33)) is still degenerate, but its coefficient matrix is more sparse and can be rapidly solved by an interior point algorithm [3]. Furthermore, the variables  $z$  play a major role in the branching strategy.

### 3.2 Branch-and-Cut

We developed a cutting plane procedure based on odd-set cuts to improve the linear relaxation bound, as suggested by Trick [14]. Padberg and Rao [11] showed that these cuts can be separated in polynomial time. They come from the fact that each round can be seen as a perfect matching in the complete graph whose node set is formed by the teams taking part in the tournament. The odd-set constraints can be described as follows. For each particular round  $k = 1, \dots, 19$ , let  $S$  be any subset of teams such that  $|S|$  is odd:

$$\sum_{i \in S, j \notin S} (x_{ijk} + x_{jik}) \geq 1 \quad \forall k = 1, \dots, 19, \quad \forall S \subseteq \{1, \dots, 20\}, \quad |S| \text{ odd.}$$

Cuts associated with odd-set constraints violated by the solution of the linear relaxation are progressively added to the enumeration tree. The number of times the linear relaxation is solved in each node is limited, because solving the linear relaxation is computationally demanding. After the linear relaxation is solved, all odd-set cuts such that

$$\sum_{i \in S, j \notin S} (x_{ijk}^* + x_{jik}^*) \leq \Delta$$

are determined by a separation procedure and appended to the model ( $x_{ijk}^*$  denotes the value of variable  $x_{ijk}$  in the optimal solution of the linear relaxation), with  $\Delta$  fixed at 0.1. This procedure is repeated while the linear relaxation bound can be improved.

The experimental results showed that this procedure strongly improved the linear relaxation bound, which was already equal to the optimal value at the root of the enumeration tree for the 2005 edition of the Chilean soccer tournament.

The branching strategy plays a major role in the success of a branch-and-cut algorithm. Branching on the  $x$  variables is not efficient, since most of them are null in integral solutions. Our branching strategy is based on the  $z$  variables. Branching on the  $x$  variables starts only after all the  $z$  variables are integral. This strategy implicitly decomposes the solution in two phases. The HAPs are computed in the first phase, while the dates of the games are established in the second. Once the  $z$  variables are fixed, the branch-and-cut algorithm needs just a few branches on the  $x$  variables to find a feasible solution or to prove infeasibility.

**Table 1.** Teams in the 2005 edition of the Chilean soccer championship

Group 1	Group 2	Group 3	Group 4
Colo-Colo (1)	Cobrelola (2)	U. de Concepción (3)	U. de Chile (0)
Audax Italiano (5)	Wanderers (6)	Unión Española (8)	U. Católica (4)
Huachipato (7)	Coquimbo (9)	Temuco (10)	Everton (11)
San Felipe (13)	Puerto Montt (12)	Palestino (16)	Cobresal (17)
Melipilla (19)	La Serena (14)	Desportes Concepción (18)	Rangers (15)

### 4 Computational Experiments

The branch-and-cut strategy described in the previous section was implemented using Visual C++ 6.0 and CPLEX 8.0. The same algorithm without the odd-set cuts was also implemented to evaluate the effectiveness of the cutting plane procedure. We refer to the first algorithm as **B&C-ANFP** and to the second as **B&B-ANFP**. The computational experiments were performed on a 3 GHz Pentium IV machine with 1 Gbyte of RAM memory. We illustrate the results obtained for the 2005 edition of the Chilean soccer championship, comparing them with those reported in [6]. Table 1 shows the name of each team and its respective group, as well as the identification used to represent each team in Tables 5 and 6.

Computation times for solving the linear programming relaxation by different algorithms available with CPLEX 8.0 are given in Table 2. Since the problem is degenerate for both the primal and dual simplex methods, their computation times were very high. Therefore, the interior point algorithm presented the best computation times for solving the linear relaxation. Table 2 shows that the new formulation considerably reduced the computation time of the interior points algorithm, leading to an efficient implementation of the cutting plane strategy.

Detailed results obtained with algorithms **B&B-ANFP** and **B&C-ANFP** are given in Table 3. For each algorithm, we report the value of the objective function, the number of nodes in the enumeration tree, and the integrality gap after some elapsed times (ranging from ten minutes to four hours). We notice that algorithm **B&B-ANFP** finds good solutions faster than **B&C-ANFP** in the beginning. However, the former was not able to find the optimal solution within a four-hour time limit. On the contrary, the cuts used by algorithm **B&C-ANFP** were able to improve the linear relaxation bound, which was already equal to the optimal value at the root of the enumeration tree. The number of nodes is much smaller for algorithm **B&C-ANFP**, that found the exact optimal solution in less than two hours of computation time.

**Table 2.** Computation times in seconds for solving the linear relaxation

Strategy	Time (s)
Primal simplex	27
Dual simplex	21
Interior points (original formulation)	12
Interior points (with the additional $z$ variables)	4

**Table 3.** Comparison between algorithms B&B-ANFP and B&C-ANFP

Elapsed time	B&B-ANFP			B&C-ANFP		
	Objective	Nodes	Gap (%)	Objective	Nodes	Gap (%)
10 min	617	140	3.6	474	120	25.9
30 min	622	600	2.9	615	330	3.9
1 h	631	1120	1.4	633	570	1.1
2 h	633	2190	1.1	640	1560	0.0
4 h	639	4860	0.2	—	—	—

**Table 4.** Comparison of algorithms B&C-ANFP and Duran et al. [6]

Algorithm	30 min	2 h
B&C-ANFP	3.9%	0.0%
Duran et al [6]	—	9.2 %

**Table 5.** Schedule provided by [6]

T\R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	<u>@11</u>	13	@9	16	@12	18	@2	6	@7	5	@8	@1	14	@19	<u>4</u>	10	@3	<u>15</u>	<u>@17</u>
1	12	@6	17	<u>@5</u>	3	@14	11	@10	2	@4	9	0	@16	8	@15	@18	<u>7</u>	<u>@13</u>	<u>19</u>
2	@8	16	@7	@3	10	@19	0	15	@1	18	@5	13	@11	<u>9</u>	17	@4	<u>14</u>	<u>@6</u>	<u>12</u>
3	5	@14	@11	2	@1	13	@9	7	@15	<u>8</u>	18	@17	<u>10</u>	@4	12	@6	0	<u>@18</u>	<u>16</u>
4	19	@12	@10	8	@18	6	@13	5	@14	1	@16	7	@9	3	<u>@0</u>	2	<u>@17</u>	<u>11</u>	<u>@15</u>
5	@3	11	@8	<u>1</u>	@9	16	@15	@4	10	@0	2	@18	17	@14	<u>19</u>	@12	6	<u>@7</u>	<u>13</u>
6	@13	1	@15	11	17	@4	10	@0	19	<u>@12</u>	<u>@14</u>	16	@15	7	@8	3	@5	<u>2</u>	<u>@9</u>
7	14	@18	2	@10	@16	15	17	@3	0	@11	12	@4	<u>19</u>	@6	<u>13</u>	9	@1	<u>5</u>	@8
8	2	@15	5	@4	11	@14	14	@19	9	<u>@3</u>	0	<u>@10</u>	12	@1	6	@13	<u>18</u>	<u>@16</u>	7
9	18	@17	0	@19	5	@10	3	@11	@8	13	@1	15	4	<u>@2</u>	16	@7	<u>12</u>	<u>@14</u>	<u>6</u>
10	17	@19	4	7	@2	9	@6	1	@5	15	@11	<u>8</u>	<u>@3</u>	@13	14	@0	<u>16</u>	@12	<u>18</u>
11	<u>0</u>	@5	3	@6	@8	12	@1	9	@13	7	10	@19	2	<u>@17</u>	18	@16	<u>15</u>	<u>@4</u>	14
12	@1	4	19	@17	0	@11	@16	13	@18	<u>6</u>	@7	<u>14</u>	@8	15	@3	5	<u>@9</u>	10	<u>@2</u>
13	6	@0	@16	15	14	@3	4	@12	11	@9	17	@2	18	10	<u>@7</u>	8	<u>@19</u>	<u>1</u>	<u>@5</u>
14	@7	3	@15	18	@13	1	@8	16	4	@17	<u>6</u>	<u>@12</u>	@0	5	@10	19	<u>@2</u>	<u>9</u>	@11
15	@16	8	14	@13	19	@7	5	@12	3	@10	18	@9	6	@12	1	<u>17</u>	<u>@11</u>	<u>@0</u>	<u>4</u>
16	15	@2	13	@0	7	@5	12	@14	17	@19	4	@16	1	<u>@18</u>	@9	11	<u>@10</u>	<u>8</u>	<u>@3</u>
17	@10	2	@1	12	@6	8	@7	18	@16	14	@13	3	@5	<u>11</u>	@2	<u>@15</u>	<u>4</u>	@19	<u>0</u>
18	@9	7	6	@14	4	@0	19	@17	12	@2	@15	5	@13	<u>16</u>	@11	1	<u>@8</u>	<u>3</u>	<u>@10</u>
19	@4	10	@12	9	@15	2	@18	8	@6	16	@3	11	<u>@7</u>	0	<u>@5</u>	@14	<u>13</u>	17	<u>@1</u>

**Table 6.** Schedule provided by B&C-ANFP

TVR	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	16	<u>@9</u>	14	<u>@3</u>	10	<u>@6</u>	13	<u>@5</u>	<u>@18</u>	1	<u>@19</u>	12	<u>@7</u>	8	<u>@17</u>	<u>11</u>	<u>@4</u>	2	<u>@15</u>
1	<u>@14</u>	15	<u>@2</u>	18	<u>@11</u>	9	<u>@12</u>	17	3	<u>@0</u>	4	<u>@10</u>	6	<u>@16</u>	<u>19</u>	<u>@13</u>	5	<u>@8</u>	7
2	<u>@5</u>	13	1	<u>@4</u>	3	<u>@15</u>	10	16	<u>@19</u>	8	<u>@18</u>	7	<u>@11</u>	17	<u>@14</u>	<u>@12</u>	6	<u>@0</u>	9
3	<u>@7</u>	14	<u>@19</u>	0	<u>@2</u>	11	<u>@17</u>	12	<u>@1</u>	<u>@13</u>	5	<u>@15</u>	4	<u>@9</u>	6	<u>@8</u>	<u>10</u>	<u>18</u>	<u>@16</u>
4	<u>@10</u>	5	<u>@16</u>	2	<u>@13</u>	19	<u>@8</u>	18	<u>@7</u>	12	<u>@1</u>	14	<u>@3</u>	<u>@6</u>	<u>15</u>	<u>@9</u>	0	<u>@17</u>	<u>11</u>
5	2	<u>@4</u>	6	<u>@10</u>	15	16	<u>@9</u>	0	<u>@12</u>	18	<u>@3</u>	8	<u>@17</u>	14	<u>@11</u>	<u>19</u>	<u>@1</u>	<u>@7</u>	<u>13</u>
6	<u>@15</u>	10	<u>@5</u>	13	<u>@16</u>	0	<u>@19</u>	8	<u>@17</u>	11	<u>@7</u>	18	<u>@1</u>	4	<u>@3</u>	<u>14</u>	<u>@2</u>	<u>@9</u>	<u>12</u>
7	3	<u>@8</u>	17	11	<u>@9</u>	12	<u>@18</u>	<u>@14</u>	4	<u>@15</u>	6	<u>@2</u>	0	<u>@10</u>	<u>13</u>	16	<u>@19</u>	5	<u>@1</u>
8	<u>@11</u>	7	<u>@9</u>	19	<u>@12</u>	<u>@17</u>	4	<u>@6</u>	14	<u>@2</u>	15	<u>@5</u>	13	<u>@0</u>	<u>10</u>	<u>3</u>	<u>@16</u>	1	<u>@18</u>
9	<u>@13</u>	0	8	<u>@17</u>	7	<u>@1</u>	5	<u>@10</u>	15	<u>@16</u>	<u>@11</u>	19	<u>@18</u>	3	<u>@12</u>	4	<u>@14</u>	6	<u>@2</u>
10	4	<u>@6</u>	<u>@15</u>	5	<u>@0</u>	13	<u>@2</u>	9	<u>@11</u>	19	<u>@17</u>	1	<u>@12</u>	7	<u>@8</u>	<u>18</u>	<u>@3</u>	<u>16</u>	<u>@14</u>
11	8	<u>@18</u>	<u>@12</u>	<u>@7</u>	1	<u>@3</u>	14	<u>@13</u>	10	<u>@6</u>	9	<u>@16</u>	2	<u>@19</u>	5	<u>@0</u>	<u>17</u>	<u>15</u>	<u>@4</u>
12	<u>@17</u>	19	<u>@11</u>	<u>@15</u>	8	<u>@7</u>	1	<u>@3</u>	5	<u>@4</u>	16	<u>@0</u>	10	<u>@13</u>	9	2	<u>@18</u>	<u>14</u>	<u>@6</u>
13	9	<u>@2</u>	18	<u>@6</u>	4	<u>@10</u>	<u>@0</u>	11	<u>@16</u>	3	<u>@14</u>	17	<u>@8</u>	12	<u>@7</u>	1	<u>@15</u>	<u>19</u>	<u>@5</u>
14	1	<u>@3</u>	<u>@0</u>	16	<u>@19</u>	18	<u>@11</u>	7	<u>@8</u>	17	13	<u>@4</u>	15	<u>@5</u>	2	<u>@6</u>	9	<u>@12</u>	10
15	6	<u>@1</u>	10	12	<u>@5</u>	2	<u>@16</u>	19	<u>@9</u>	7	<u>@8</u>	3	<u>@14</u>	18	<u>@4</u>	<u>@17</u>	<u>13</u>	<u>@11</u>	0
16	<u>@0</u>	17	4	<u>@14</u>	6	<u>@5</u>	15	<u>@2</u>	13	9	<u>@12</u>	11	<u>@19</u>	1	<u>@18</u>	<u>@7</u>	8	<u>@10</u>	<u>3</u>
17	12	<u>@16</u>	<u>@7</u>	9	<u>@18</u>	8	3	<u>@1</u>	6	<u>@14</u>	10	<u>@13</u>	5	<u>@2</u>	0	<u>15</u>	<u>@11</u>	4	<u>@19</u>
18	<u>@19</u>	11	<u>@13</u>	<u>@1</u>	17	<u>@14</u>	7	<u>@4</u>	0	<u>@5</u>	2	<u>@6</u>	9	<u>@15</u>	<u>16</u>	<u>@10</u>	12	<u>@3</u>	8
19	18	<u>@12</u>	3	<u>@8</u>	14	<u>@4</u>	6	<u>@15</u>	2	<u>@10</u>	0	<u>@9</u>	16	11	<u>@1</u>	<u>@5</u>	7	<u>@13</u>	17

In Table 4, we compare the results obtained by algorithm B&C-ANFP with those obtained by the strategy proposed in 6. We give the relative integrality gap from the optimal solution after 30 minutes and after two hours of computation time (on a 2.4 GHz Pentium IV computer for 6) for both algorithms. Algorithm B&C-ANFP not only found a better solution quickly, but also found a much better – and optimal – solution after the same time the approach in 6 took to find a solution 9.2% away from the optimal value.

The schedules provided by 6 and B&C-ANFP are presented in Tables 5 and 6, respectively. The lines correspond to teams and the columns to rounds. Games between teams from the same group are underlined. Table 5 shows that the schedule obtained by 6 has games between teams from the same group spread along all rounds, while the schedule provided by the new algorithm has all games between teams from the same group in the last five rounds of the tournament.

## 5 Concluding Remarks

We proposed a new formulation for the highly constrained Chilean soccer tournament scheduling problem. Valid inequalities were derived and appended to the formulation to improve its linear relaxation bound. A branching strategy based on the new variables was used to speedup convergence.

The new formulation considerably reduced the computation times needed to solve the linear relaxation. The odd-set cuts improved the linear relaxation bound, which was already equal to the optimal value at the root of the enumeration tree. The new algorithm B&C-ANFP significantly outperformed the previous approach and found the optimal solution in less than two hours. Future work will deal with new constraints and objective functions imposed by TV sponsors, as well as with heuristics for providing integer feasible solutions for the model.

## References

1. Bartsch, T., Drexl, A., Kroger, S.: Scheduling the professional soccer leagues of Austria and Germany. *Computers and Operations Research* 33, 1907–1937 (2006)
2. Bean, J.C., Birge, J.R.: Reducing traveling costs and player fatigue in the National Basketball Association. *Interfaces* 10, 98–102 (1980)
3. Bertsimas, D., Tsitsiklis, J.N.: *Introduction to Linear Optimization*. Athena Scientific, Nashua, NH (1997)
4. Costa, D.: An evolutionary tabu search algorithm and the NHL scheduling problem. *Information Systems and Operational Research* 33, 161–178 (1995)
5. Della Croce, F., Oliveri, D.: Scheduling the Italian football league: An ILP-based approach. *Computers and Operations Research* 33, 1963–1974 (2006)
6. Duran, G., Guajardo, M., Miranda, J., Sauré, D., Souyris, S., Weintraub, A., Carmash, A., Chaineu, F.: Programación matemática aplicada al fixture de la primera división del fútbol Chileno. *Revista Ingeniería de Sistemas* 5, 29–46 (2005)
7. Goossens, D., Spieksma, F.C.R.: Scheduling the Belgian soccer league. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 420–422 (August 2006)
8. Henz, M.: Scheduling a major college basketball conference revisited. *Operations Research* 49, 163–168 (2001)
9. Henz, M., Müller, T., Thiel, S.: Global constraints for round robin tournament scheduling. *European Journal of Operational Research* 153, 92–101 (2004)
10. Nemhauser, G.L., Trick, M.A.: Scheduling a major college basketball conference. *Operations Research* 46, 1–8 (1998)
11. Padberg, M.W., Rao, M.R.: Odd minimum cut-sets and b-matchings. *Mathematics of Operation Research* 7, 67–80 (1982)
12. Ribeiro, C.C., Urrutia, S.: Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006*. LNCS, vol. 3867, pp. 149–159. Springer, Heidelberg (2007)
13. Trick, M.A.: A schedule-and-break approach to sports scheduling. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 242–253. Springer, Heidelberg (2001)
14. Trick, M.A.: Integer and constraint programming approaches for round robin tournament scheduling. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 63–77. Springer, Heidelberg (2003)
15. Zhang, H.: Generating college conference basketball schedules by a SAT solver. In: *Proceedings of the 5th International Symposium on the Theory and Applications of Satisfiability Testing*, Cincinnati, pp. 281–291 (2002)

# **Course Timetabling**



# Modeling and Solution of a Complex University Course Timetabling Problem

Keith Murray<sup>1</sup>, Tomáš Müller<sup>1</sup>, and Hana Rudová<sup>2</sup>

<sup>1</sup> Space Management and Academic Scheduling, Purdue University  
400 Centennial Mall Drive, West Lafayette, IN 47907-2016, USA  
{kmurray,muller}@purdue.edu

<sup>2</sup> Faculty of Informatics, Masaryk University, Botanická 68a,  
Brno 602 00, Czech Republic  
hanka@fi.muni.cz

**Abstract.** The modeling and solution approaches being used to automate construction of course timetables at a large university are discussed. A course structure model is presented that allows this complex real-world problem to be described using a classical formulation. The problem is then tackled utilizing a course timetabling solver model that transforms it into a constraint satisfaction and optimization problem. The tiered structure of this approach provides flexibility that is helpful in solving the multiple subproblems that arise from decomposition of the university-wide problem. A production system has been partially implemented and results of early use are presented. Practical issues raised during the implementation of the automated timetabling system are also discussed.

## 1 Introduction

Timetabling is a widely studied area and many potentially useful algorithms have been offered for solving the university course timetabling problem, as evidenced by several recent surveys [7,16,19]. Unfortunately, much of the work in this area has been conducted using artificial data sets or based on actual problems that have been greatly simplified. Methods developed have also rarely been extended to the solution of actual university problems of any large scale. McCollum offers a good review of this situation in [11].

The major differences between many of the problems studied and their real-life counterparts are the additional complexity imposed by course structures, the variety of constraints imposed, and the distributed responsibility for information needed to solve such problems at a university-wide level. University timetabling problems may also involve the solution of multiple subproblems with very different characteristics. In practice, therefore, the solution process should not be specifically tailored to a single problem type.

The work described here has been motivated by the need to create and modify course timetables at Purdue University that better meet student course demand and allow students to be assigned to the constituent course sections in a way

that minimizes conflicts. Purdue is a large (39,000 students) public university with a broad spectrum of programs at the undergraduate and graduate levels. In a typical term there are 9,000 classes offered using 570 teaching spaces. Approximately 259,000 individual student class requests must be satisfied. The complete university timetabling problem is decomposed into a series of subproblems to be solved at the academic department level, where the resources required to provide instruction are controlled. Several other special problems, where shared resources or student interactions are of critical importance, are solved institution wide. A major consideration in designing the system has been supporting distributed construction of departmental timetables while providing central coordination of the overall problem. This reflects the distributed management of instructional resources across multiple departments at the University. The general definition of the university-wide timetabling problem described in this paper is similar to the problem studied by Carter [6] at the University of Waterloo, and the influence of that work can be seen here, though the solution methods used differ significantly. It is hoped that the results of the present work will, likewise, be beneficial to other institutions seeking to improve their ability to construct course timetables for their students.

This paper discusses the approach used for modeling and solving the additional complexities involved in developing automated solution techniques for a real-life course timetabling problem on the scale of a large university. Although a specific example is discussed, many of the methods used should be applicable to modeling and solving other complex problems.

The complexity of the university course timetabling problem studied here has been broken down by developing a logical data model allowing all courses to be represented as hierarchical groupings of classes with additional parent-child relationships and constraints governing their placement. This allows use of one standard class-oriented problem formulation rather than having to develop different models and solution methods to work with the wide variety of ways that departments organize the instruction given in their courses.

A flexible and general solution technique has been developed for solving course timetabling problems and applied to all of the departmental and special subproblem. Rather than applying multiple solution methods, each optimized around the characteristics of a specific problem, a single solution approach allows the outcomes of all of these subproblems to be easily combined into a complete solution and facilitates optimization of the sectioning of students across the complete timetable.

Each of these topics will be explored in greater depth in the next three sections of this paper, followed by observations on creating a general framework that is very useful in developing a practical solver. Several practical issues faced while implementing a real-world system are then discussed, including competitive behavior among users, making changes to solutions, and managing data consistency. Some results from actual use of the system to solve departmental and campus problems are also presented. In addition, links are provided to the problem data used in this work to promote further study by other researchers using real data instances.

## 2 Problem Decomposition

Timetabling is a resource allocation problem; therefore, at most universities responsibility for constructing the timetable is distributed among the academic units with the faculty, physical facilities, and other resources required for offering instruction. Providing support for this distributed responsibility is important because departmental timetablers have a much more intimate knowledge of the needs of the courses offered, the faculty who might be able to teach a particular class, and the spaces available for specialized instruction than any database that might be maintained centrally. Maintaining each department's sense of ownership in the timetables that are produced is also an important factor in their acceptance of the solutions produced by an automated timetabling process. The process needs to be one that assists them rather than replaces them.

Student degree programs may or may not consist of courses offered primarily within a single academic unit. If they do, the larger university timetabling problem may be decomposed into a series of departmental problems with little likelihood of creating class time assignment conflicts for students. A number of individual departmental or school level problems have been studied in the literature [15,9,17]. The problem becomes more complex, however, when students attend courses from multiple academic units and the solutions are dependent upon the availability of students for the classes across multiple problems. Here the overall problem can not be easily decomposed along the lines of academic units and additional coordination is required.

In the case of Purdue University, there are many large introductory courses that serve students in almost all degree programs. Since there are substantial numbers of students enrolled in more than one of these courses, they create a large dependency between the timetables of individual departments offering instruction. To deal with this, the cluster of large courses with students from many disciplines is split off as a separate problem that is solved by a central scheduling office with input from departmental timetablers. This problem is solved first, assigning times in a set of centrally managed lecture facilities, so that the results are available to all of the departmental timetablers as they solve the remainder of their problems. Most of the courses in the departmental problems primarily serve students in programs offered by that department. There are other groupings that occur however. There are several colleges where four or five departments create their timetable together because all of the departments provide courses that serve the same degree program. There are also several special problems, such as assigning classes that require shared campus computer laboratories. The ability to solve all of these different types of problems is one of the major challenges in automating the timetabling process at a large university.

To better understand the effect on solution quality of decomposing the problem into multiple parts, some post-timetabling experiments were run solving all of the parts together as a single problem. Although data for all departmental and other special problems at the University is not yet available, comparison of several parts addressed separately and combined as a whole gives some indication of what is lost in solution quality as a result of the decomposition. Table 1 shows

**Table 1.** Combined solution properties of four problems (Spring 2007)

Test case	Final	Run separately	Run combined
Assigned variables	1756.0	$1756.0 \pm 0.0$	$1756.0 \pm 0.0$
Time [min]	–	$31.8 \pm 5.9$	$44.1 \pm 9.7$
Student conflicts	1258	$907.7 \pm 25.1$	$849.9 \pm 27.4$
Preferred time [%]	86.2	$90.6 \pm 0.7$	$91.2 \pm 1.1$
Preferred room [%]	82.8	$83.7 \pm 0.5$	$83.9 \pm 0.4$
Preferred distribution [%]	61.0	$66.4 \pm 3.7$	$71.6 \pm 3.4$

a summary of several measures of solution quality for the large lecture problem, the central computing labs, and two different departmental problems when run separately and when combined. (Average values and RMS (root-mean-square) variances between the best solutions found for 10 different runs are presented. Run time is 30 minutes for individual problems and 120 minutes for the combined problem using 2.13 GHz Pentium M, Java 1.5.0, 2GB RAM.) The same room resources were used for the separate and combined runs. As expected, there is a decrease in the number of student conflicts when all of the problems are considered together. There is also a slight improvement in the overall satisfaction of time, room, and distribution preferences. These improvements must be considered as theoretical, however, since the combined solution has not been reviewed and accepted by all of the schedule managers who have the real final say on solution quality.

The column labeled *final* summarizes these same measures of solution quality for the actual final timetables produced by University schedule managers using this application for Spring 2007. These solutions were initially computed individually using the automated solver (see Section 4); however, some additional changes were applied manually later in the process using the solver in its interactive mode (see Section 6.2).

## 2.1 Interactions Between Problems

As described in the previous section, the Purdue University timetabling problem is naturally decomposed into

- a centrally timetabled large lecture room problem (about 800 classes timetabled into 55 rooms with sizes up to 474 seats),
- individually timetabled departmental problems (about 70 problems with 10 to 500 classes using departmental laboratory spaces and centrally managed classrooms allocated to departments based on expected class hours),
- and a centrally timetabled computer laboratory problem (about 450 classes timetabled into 36 rooms with 20 to 45 seats).

The large lecture room problem consists of the largest classes on campus that are attended by students from multiple departments. This problem is also very dense. On average, rooms are utilized over 70% of the available time, and this

rate increases with room size (utilization is over 85% for all rooms above 100 seats and about 97% for the four largest rooms having over 400 seats). Since there are many interactions between this problem and the departmental problems, the large lecture problem is solved first and the departmental problems are solved on top of this solution.

On the opposite end of the spectrum, the computer laboratory problem is solved at the very end of the process, on top of the large lecture room and departmental problem solutions. It contains only small classes, most of which have many sections (laboratories are normally the smallest subparts of a course). A typical example is a course having one large lecture class for 100 students, two departmental recitations with 50 students each, and four computer laboratories of 25 students.

The departmental problems are solved more or less concurrently. These problems are usually quite independent of one another, occurring in mostly different sets of rooms, with separate instructors and students. However, there are some cases with higher levels of interaction, particularly among students. In order to address these situations, a concept referred to as ‘committing’ solutions has been introduced. Each user of the timetabling system (e.g., a departmental schedule manager) can create and store multiple solutions. At the end of the process a single solution must be selected and committed. During the commit, all conflicts between the current solution and all other solutions that have already been committed are checked and the commit is successful only when there are no hard conflicts between these solutions. Each problem being solved also automatically considers all of the previously committed solutions. This means that a room, an instructor, or a student is available at a particular time only if that time is not already occupied in a committed solution for a different problem. This approach can be beneficial, for instance, in a case where there are two or more departments with many common students. Here, the problems can be solved in an agreed upon order (the second department will solve its problem after the first department commits its solution). Moreover, if a room must be shared by two departments, a room-sharing matrix can be defined, stating the times during the week that a room is available for each department to use. Finally, there is also an option to combine two or more individual problems and solve as one larger problem, considering all of the relations between the problems in real time.

## 2.2 Problem Characteristics

Each of the problems that the overall university timetabling problem has been decomposed into has characteristics that are different from many of the other problems. Some of the different attributes of the large lecture room problem (LLR), computer laboratory problem (LAB) and two selected departmental problems (D1, D2) are listed in Table 2.

If solved independently, the large lecture room problem is the most difficult. In addition to being the largest problem in terms of number of classes, it must consider more students requesting multiple classes within the problem, rooms with a greater variation in size, very high utilization in the larger rooms, and large

**Table 2.** Characteristics of selected problems (Spring 2007)

Problem	LLR	D1	D2	LAB
Number of classes	804	440	69	442
Avg. number of classes per type of instruction	1.25	3.52	1.50	4.8
Avg. number of hours per class	2.40	2.43	2.30	1.97
Avg. number of meetings per class	2.09	2.32	1.67	1.25
Avg. number distribution constraints per class	0.68	2.94	0.78	1.82
Number of rooms	55	25	6	36
Room sizes	40–474	24–51	14–48	20–45
Avg. room utilization [hours/week]	35.0	42.8	26.5	24.2
Average distance between rooms [m]	223.9	83.9	21.5	159.7
Number of students	27881	11992	1312	8408
Avg. number of classes per student	3.15	1.11	1.40	1.14
Classes with an instructor assigned [%]	69.8	33.9	60.9	13.35
Avg. number of classes per instructor	1.25	1.49	1.68	2.11

distances between some rooms. There are also fewer alternatives for sectioning student enrollments. The average number of classes per type of instruction offered as part the course (e.g., lecture, laboratory) is only 1.25. For departmental problems, besides the properties listed in Table 2, it is also necessary to consider that they are being built on top of the large lecture problem and that there are many teachers and students in common between them and the LLR problem. This can be an even greater complication for the computer laboratory problem, since it is being built on top of all of the other problems.

### 3 Modeling the University Course Timetabling Problem

Arguably, the biggest obstacle to solving actual university course timetabling problems is that the complexity can increase considerably beyond that represented in standard formulations of the problem [7,16,19]. As the complexity increases, it is easy to be caught in the dual bind that the problem is both more challenging to develop an effective solution approach for, and this approach is less likely to be usable on other university timetabling problems, or even on all problems that may exist at a single institution. This complexity arises from many aspects of real-life problems. Among the most important are the structure of course offerings and the wide range of constraints that arise. As noted above, student enrollments across disciplines and shared use of resources between autonomous departments are also of concern. Other factors include the number and uniformity of the meeting times, dates, and locations over which classes need to be assigned.

Instructional Offering <i>Alternate listing</i>	Configuration	Subpart <i>Parent</i> <i>Child</i>	Class  <i>multiple offerings...</i>
MA 100 – Calculus ENGR 101	Traditional	Lecture	Lec 1    Lec2
		Recitation	Rec1    Rec4 Rec2    Rec5 Rec3    Rec6
	Computer-aided	Lecture	Lec3
		Recitation	Rec7 Rec8
		Laboratory	Lab1 Lab2

**Fig. 1.** Model of course structure. Example shows representation of an instructional offering with two catalog listings, two alternate configurations, and two subparts linked by a parent–child relationship.

### 3.1 Course Structure

The structure of course offerings may be the most problematic. Although the problem is generally labeled ‘course timetabling’, in actuality it is the individual classes that make up a course which must be timetabled. In this paper, a class is defined to be a series of similar meetings for a subset of students enrolled in the course. Courses are usually composed of multiple classes that may be known as lectures, tutorials, laboratories, etc. Students normally enroll to various course offerings that are required to meet the requirements of their degree program, and most university information systems are organized around courses (known as modules in some regions) as the unit of instruction.

The modeling problem becomes one of creating a logical data structure that can be used to translate all parts of the course structure, and the relationships between these parts, into a set of classes and an extended set of constraints between them. The complex real-life problem can then be solved at the class level using a standard formulation of the course timetabling problem.

To incorporate all of the course structures found in the Purdue problem, a four-level model was developed that breaks down each course into as many as four tiers to reflect the relationships among all of the classes that constitute it. While in a simple lecture course the class and the instructional offering are one and the same, for a large course there may be tens or hundreds of classes associated with a single instructional offering. A diagrammatic representation of the course structure model and an example showing the parts of a more complex course is shown in Figure 1. A more detailed description of each layer in this model is given in the paragraphs below.

For the sake of clarity, the term *instructional offering* has been used in this model to distinguish the highest level of the structure from *course*, which is the more usual term for a series of lessons containing the subject matter to be





















taught. This was necessary since many universities have adopted the habit of listing the same subject matter in their course catalogs under more than one subject and course number. This boutique naming system can create quite a complication when attempting to automate timetabling since it results in many courses requiring the same times in the same rooms with the same instructors. It is also difficult under this system to know how many students are actually being taught together. The model addresses such cases by treating all course identifiers as pseudonyms and linking the courses together to form a single instructional offering. Naturally, all courses that are linked together must have the same structure. The *alternate listing* below instructional offering in Figure 1 indicates this linkage with other courses that actually meet together as part of a single instructional offering. The instructional offering is the basic organizing unit in this model. The offering is then divided into its constituent classes, which are the unique entities to be timetabled.

Many courses (or instructional offerings in the model's terminology) have multiple subparts, such as tutorials or laboratories, that are associated with a parent lecture. In some cases, the course may even be offered with different configurations of these subparts. One instructor, for example, may wish to teach the course material entirely in a lecture format whereas another may wish to devote one day per week to small discussion groups. The *configuration* level is included in this model to account for these types of situations.

At the *subpart* level, the type of instruction usually takes on different characteristics. The students in the course may be divided into smaller groups for different activities and other types of facilities may be required. It may or may not be important for one group of students to be together in two or more different subparts of a course, such as wanting all students sectioned into a discussion group to also be in a laboratory together. To accommodate such needs, a parent-child relationship has been included in the model at the subpart level. If a parent-child relationship is established between two subparts, all students in a class belonging to the child subpart must also be sectioned to the appropriate parent class. Constraints are generated prohibiting an overlap in time between parent and child classes. In the GUI built for entering data, these parent-child relationships are set up much like file folders in a directory tree as indicated by the indented subparts in Figure 1. Any attributes or preferences that apply to all classes within a subpart can be set at the subpart level. An illustration of how the structure is displayed by the interface is shown in Figure 2.

Timetabling takes place at the *class* level. There will typically be multiple classes associated with each subpart, especially when tutorial or laboratory sections are involved. Each class inherits attributes and preferences set on the subpart level, or these may be set for an individual class. Attributes or preferences set at the class level will override those set at higher levels. Each class must indicate the amount of time it meets, desired meeting pattern, weeks the class should meet during the term, and facility needs. Specific time, room, and room feature preferences or requirements may also be set. If an instructor is entered, preferences may also be inherited from those set on the instructor.



	Mins Per		Limit	Manager	Date Pattern	Time Pattern	Preferences			Instructor
	Demand	Week					Room	Time	Distribution	
<b>ME 263</b>										
ME 263H										
Lecture										
		150	96	LLR	Full Term	3 x 50 2 x 75	 WTHR  Computer			
	Recitation	100	96	ME	Full Term	2 x 50	 ME 120  ME 236  Classroom			
	Laboratory	50	84-120	LAB	Even Wks	1 x 50	 Windows XP			
Lec 1		150	96	LLR	Full Term	3 x 50 2 x 75	 WTHR  Computer			J. Smith C. Bing
Rec 1		100	48	ME	Full Term	2 x 50	 ME 120  ME 236  Classroom	Back-To-Back		J. Novak
	Lab 1	50	14-20	LAB	Even Wks	1 x 50	 Windows XP			
	Lab 2	50	14-20	LAB	Even Wks	1 x 50	 Windows XP			
	Lab 3	50	14-20	LAB	Even Wks	1 x 50	 Windows XP			
Rec 2		100	48	ME	Full Term	2 x 50	 ME 120  ME 236  Classroom	Back-To-Back		J. Novak
	Lab 4	50	14-20	LAB	Odd Wks	1 x 50	 Windows XP			
	Lab 5	50	14-20	LAB	Odd Wks	1 x 50	 Windows XP			
	Lab 6	50	14-20	LAB	Odd Wks	1 x 50	 Mac Os X			

**Fig. 2.** Structure of classes as displayed in user interface. The configuration is displayed in the gray shaded area. Individual classes to be timetabled are listed below.

### 3.2 Constraints

The large number and variety of constraints that arise, both from the class structure and special requirements, also adds to the difficulty in finding a solution to real problems. In addition to the usual constraints specifying that an instructor can only teach one class at a time, there can only be one class per room, and the room must accommodate class requirements, each of the departmental and special problem instances has additional hard and soft constraints that differ according to the concerns of the individual unit. This imposes a demand that the solution method must be very robust so that it can accommodate each of these different problem formulations.

Each schedule manager is able to set whatever hard or soft constraints are considered necessary on the problem he/she is responsible for. These fall into the categories listed in Table 3. A consistent scale (required, strongly preferred, preferred, neutral, discouraged, strongly discouraged, prohibited) is established for setting all of these constraints. Required and prohibited indicate hard constraints. Distribution constraints may be set between individual classes or between all classes associated with an instructional offering or subpart. Managers may also override the normal hard constraint requiring one class per room by setting the number of rooms an individual class should be timetabled to.

**Table 3.** User set constraints

Time:	Meeting Time Pattern Individual Times
Rooms:	Specify Individual Buildings/Rooms Specify User Defined Room Group
Room Features:	Select Based on User Defined Set of Features
Class Distribution:	Time Between Time Order of Classes Place Classes in Time Groups Use Same Meeting Dates/Times Spread in Time Restrict Classes Meeting at Same Time Room Sharing

As noted in the section on course modeling above, a number of additional constraints are automatically set on each problem due to the structure of the course. These require a student to be sectioned into one class for each subpart and prohibit conflicts between parent and child classes. Time assignments for all classes within a subpart are also automatically spread across all non-prohibited times. In addition, an automatic calculation of distances between rooms is performed to penalize class placements that require students or instructors to travel large distances between consecutive classes. There is also a set of constraints that seek to ensure efficient use of resources by discouraging use of larger rooms than required or time placements that leave gaps in the schedule that are inconsistent with the standard time patterns used by most classes.

## 4 Solution Methods

The solver applied to this problem is based on constraint satisfaction techniques [8] which are frequently applied to solve timetabling problems [9,18,5]. A constraint satisfaction and optimization problem (CSOP) consists of a set of variables having finite domains, a set of (hard) constraints restricting the values that these variables can be assigned at the same time, and an objective function. In a complete solution to a CSOP, a value is assigned to every variable such that every hard constraint is satisfied. The objective can be expressed by the soft constraints and the aim is to find a complete solution that violates the least number of these soft constraints (or a weighted sum of violated soft constraints).

Since a large majority of classes meet in a regular fashion, it is normally possible to represent all meetings of a class using a single variable. Although not required by the solver, tying meetings together into standard time patterns in this way considerably simplifies the problem constraints. As a result of modeling classes as a homogeneous series of events, most classes have all meetings in the same room, taught by the same instructor, and at the same time of day. Meetings are also separated by a uniform number of days. A typical standard time pattern for a class meeting 3 hours per week is to meet 3 days per week

for 1 hour. Moreover, in the time patterns used by Purdue, these three meetings can only be on Monday, Wednesday, and Friday starting at half past each hour.

All valid placements of a course in the timetable have a one-to-one correspondence with values in the variable's domain. This means that each value encodes the selected date pattern (weeks when the class is to be taught), time pattern, and starting time. Each value also encodes the instructor and the given number of meeting rooms. Additionally, each such placement also encodes preferences (soft constraints), combined from the preferences for time, room, and room features, inherited from various levels of the input data. Only placements with valid times and rooms are present in a class's domain. For example, if an instructor computer (room feature) is required, only placements in a room containing a computer are present. Also, only rooms large enough to accommodate all the enrolled students can be present in valid class placements. Similarly, if a time interval is prohibited, no placement containing this time interval is in the class's domain.

The variable and value encodings described above leave only two types of hard constraints to be implemented: hard resource constraints (e.g., only one class can be taught by an instructor, or in a room, at any time, and only when that resource is available), and hard distribution constraints (expressing required or prohibited relations between several classes: e.g., that two sections of the same lecture can not be taught at the same time, or that a classes must be taught after another). There are three types of soft constraints. The first category of soft constraints are those on times and rooms. The second group of soft constraints is formed by student requirements. Each student can enroll in several classes, so the aim is to minimize the total number of student conflicts among these classes. Finally, there are soft distribution constraints that express preferred or discouraged relations between groups of classes.

## 4.1 Timetabling Solver

The solver is based on an iterative forward search algorithm [13,15]. This algorithm is similar to local search methods; however, in contrast to classical local search techniques, it operates over feasible, though not necessarily complete, solutions. In these solutions some classes may be left unassigned. All hard constraints on assigned classes must be satisfied however. Such solutions are easier to visualize and more meaningful to human users than complete but infeasible solutions. Because of the iterative character of the algorithm, the solver can also easily start, stop, or continue from any feasible solution, either complete or incomplete. Moreover, the algorithm is able to support dynamic aspects of the minimal perturbation problem [4,15], allowing the number of changes to the solution (perturbations) to be kept as small as possible.

The search is processed iteratively (see Figure 3 for the algorithm). During each step, a variable is selected. Typically an unassigned variable is chosen. An assigned variable may be selected when all variables are assigned but the solution is not good enough: e.g., when there are still many violations of soft constraints. Once a variable is selected, a value from its domain is chosen for assignment. Even if the 'best' value is selected, its assignment to the selected

```

procedure SOLVE(initial)           // initial solution is the parameter
  iteration = 0;                   // iteration counter
  current = initial;               // current solution
  best = initial;                  // best solution
  while canContinue(current, iteration) do
    iteration = iteration + 1;
    variable = selectVariable(current);
    value = selectValue(current, variable);
    UNASSIGN(current, CONFLICTING_VARIABLES(current, variable, value));
    ASSIGN(current, variable, value);
    if better(current, best) then best = current
  end while
  return best
end procedure

```

**Fig. 3.** Pseudo-code of the search algorithm

variable may cause some hard conflicts with already assigned variables. Such conflicting variables are removed from the solution and become unassigned. Finally, the selected value is assigned to the variable. The algorithm attempts to move from one (partial) feasible solution to another via repetitive assignment of a selected value to a selected variable. During this search, the feasibility of all hard constraints in each iteration step is enforced by removing conflicting variables. The search is terminated when the desired solution is found or when there is a timeout. The best solution found is then returned.

Application of this algorithm to the course timetabling problem has been described previously [15] in greater detail; however, its use is not limited to course timetabling. The algorithm can be easily applied to various constraint satisfaction and optimization problems and has been extended in several ways [13]. An example is the use of a learning technique called conflict-based statistics that has been developed to improve the quality of the final solution [14]. In this approach, conflicts during the search are memorized in order to minimize their potential repetition.

## 4.2 Sectioning

Many course offerings consist of multiple classes, with students enrolled in the course divided among them. These classes are often linked by a set of constraints, namely:

- Each class has a limit stating the maximum number of students who can be enrolled in it.
- A student must be enrolled in exactly one class for each subpart of a course.
- If two subparts of a course have a parent–child relationship, a student enrolled in the parent class must also be enrolled in one of the child classes.

Moreover, some of the classes of an offering may be required or prohibited for certain students, based on reservations that can be set on an offering, a configuration, or a class.

Before implementing the solver, an initial sectioning of students into classes is processed. This sectioning is based on Carter's [6] homogeneous sectioning and is intended to minimize future student conflicts. However, it is still possible to improve on the number of student conflicts in the solution. This can be accomplished by moving students between alternative classes of the same course during or after the search. Several approaches have been discussed in the literature on the sectioning subproblem [2,3,10], usually incorporating some iteration between sectioning and timetabling during the solution process.

In the current implementation, students are not re-sectioned during the search, but a student re-sectioning algorithm is called after the solver is finished or upon the user's request. The re-sectioning is based on a local search algorithm where the neighboring assignment is obtained from the current assignment by applying one of the following moves:

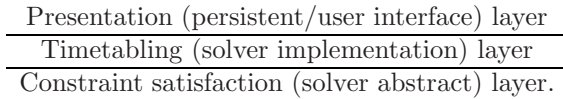
- Two students enrolled in the same course swap all of their class assignments.
- A student is re-enrolled into classes associated with a course such that the number of conflicts involving that student is minimized.

The solver maintains a queue, initially containing all courses with multiple classes. During each iteration, an improving move (i.e., a move decreasing the overall number of student conflicts) is applied once discovered. Re-sectioning is complete once no more improving moves are possible. Only consistent moves (i.e., moves that respect class limits and other constraints) are considered. Any additional courses having student conflicts after a move is accepted are added to the queue.

Since students are not re-sectioned during the timetabling search, the computed number of student conflicts is really an upper bound on the actual number that may exist afterward. To compensate for this during the search, student conflicts between subparts with multiple classes are weighted lower than conflicts between classes that meet at a single time (i.e., having student conflicts that cannot be avoided by re-sectioning).

## 5 General Framework for Modeling and Solving Problem

One observation that can be made as a result of the work modeling the university course timetabling problem and developing a solution method capable of addressing the level of complexity involved, is that the overall approach to structuring the problem is a critical factor in achieving a successful outcome. Solving this problem has required multiple iterations, each expanding the course data structure and the constraints needed to accurately represent the problem. Without having separate components to the solution process, each able to address specific aspects of the overall problem without requiring extensive rework of other components, it would have been much more difficult to accommodate all of the necessary changes. The structure that has been developed in the course of solving the problem described here can be modeled using the following three-tier architecture:



The *presentation* layer consists of data persistency, business logic and a user interface for data entry and operation of the timetabling solver. It contains the structure of courses, classes, rooms, instructors, constraints, preferences, and requirements as entered by users. All data on this level are persistent, stored in the database in a similar structure as it is presented to the users. This layer also contains timetable solutions in the structure that they are stored in the database and presented to the users.

The *timetabling* layer contains the data model being used by the timetabling solver. There are no persistent data in this layer and there is no direct communication between this layer and the database or the user interface. Communication between (the business logic of) the *presentation* layer and the *timetabling* layer consists of two parts. The first intermediates the interaction between the data model of the *presentation* layer and the *timetabling* layer: i.e., loading the data into the solver and saving the resulting solution from the solver. There are various transformations present in this interface. For instance, the entire course structure is transformed into classes and all preferences are inherited to the class level. The second interface allows the user interface from the *presentation* layer to operate the solver as well as to present the solution from the *timetabling* layer to the user. The data model of this structure contains classes, room and time assignments, room, instructor, distribution, and other constraints as well as the problem-specific heuristics that are used to guide the solver.

The *constraint satisfaction* layer consists of an implementation of the constraint satisfaction and optimization solver, working with variables, values and (hard and soft) constraints. The solver is guided by a general set of heuristics (e.g., variable and value selection criteria) without knowledge of any classes, rooms or other timetabling specific primitives. The interface between the *constraint satisfaction* and *timetabling* layers is implemented through general abstract objects like variables, values, and constraints by the corresponding timetabling problem specific objects like classes, time/room assignments, and resource or distribution constraints. Similarly, some of the general heuristics are extended by the problem specific ones of the *timetabling* layer.

The aim of this architecture is to be able to alter one of the layers without having to change the others. In this way the solver can be modified, or even completely changed, without any changes being made to the upper layers. Similarly, most changes to the user interface or the database structure can end at the interface that is transforming data models between the *presentation* and the *timetabling* layers.

## 6 Practical Issues Arising During Implementation

In the course of developing a system that is usable in practice, it was necessary to confront a number of issues that are not typically addressed in the literature on

timetabling, but which are critical to successful implementation. These included issues of the ‘fairness’ of a solution across all departments with classes being timetabled, the ease of introducing changes after a solution has been generated, and the ability to check and resolve inconsistencies in input data.

## 6.1 Competitive Behavior

A complicating aspect of real timetabling problems is that there is competition for preferred times and rooms. Hard and soft constraints placed on the problem are often reflective of this competitive behavior (e.g., limited instructor time availability, restrictive room requirements).

Hard constraints limit the solution space of the problem to reflect the needs or desires of those who place them. Soft constraints introduce costs into the objective function when violated. In either case, the more constraints placed on the problem by a particular class, instructor, or class offering department, the greater influence they will have on the solution. The general effect is to weight the solution in the favor of those who most heavily constrain the problem. This can create both harder problems to solve and solutions that are perceived as unfair by other affected groups or individuals. Inequity in the quality of time and room assignments received by different departments and faculty members doomed a previous attempt at automating the timetabling process at Purdue [12].

To counteract the tendency of the solution to favor those who place the most restrictions, a number of market leveling techniques were employed while modeling and solving the problem. The first was to weight the value of time preferences inversely proportional to the amount of time affected. A class with few restrictions on the times it may be taught has those restrictions more heavily weighted than a class with many restrictions. The intent is to make the total weight of all time restrictions on any class roughly equal. A second technique used in the solver was to introduce a balancing constraint. This is a semi-hard constraint in that it initially requires the classes offered by each department to be spread equitably across all times available for the class, but is automatically relaxed to become a cost penalty for poorly distributing time assignments if the desired distribution is overly constraining. Addressing this aspect of the real-world problem was a key component of gaining user acceptance.

## 6.2 Interactive Changes

While it was known early that it would be necessary to deal with changes after an initial solution was found, it became clear the first time the system was used in practice that an interactive mode for exploring the possibility of changes, and easily making them, would be necessary. Following the philosophy of wanting to minimize the number of changes needed to a solution [4,15], an approach was developed to present all feasible solutions (and their costs) that can be reached via a backtracking process of limited depth. The user is allowed to make the determination of the best tradeoff between accommodating a desired change and the costs imposed on the rest of the solution with a knowledge of what those

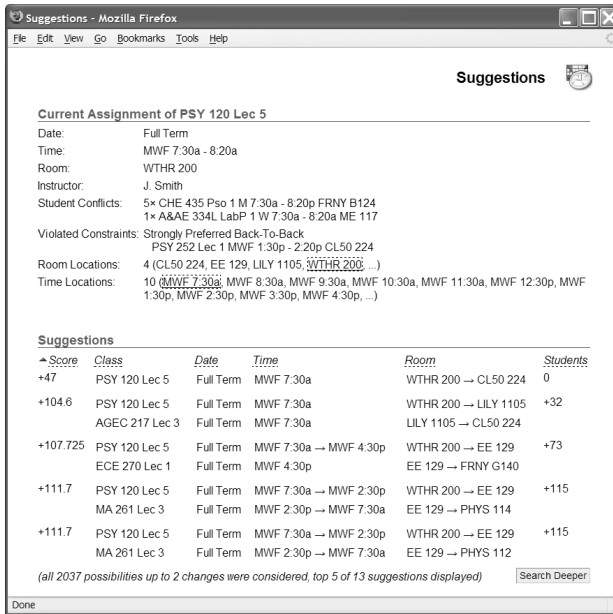


Fig. 4. User interface showing a list of suggestions provided to the user for a class

costs will be. A further refinement was to allow some of the hard constraints to be relaxed in this mode. This means, for instance, that the user can put a class into a room different from the ones that were initially required.

Figure 4 displays a list of suggestions (nearby feasible solutions) for reassigning a selected class. The user may either pick one of these alternative solutions, ask the solver to provide additional suggestions by increasing the search depth (changes in up to two class placements are allowed by default), or assign the class manually by selecting one of the possible placements. In this last case, a list of conflicting classes is shown together with a list of suggestions for resolving these conflicts. The user may either apply the selected assignment (which will cause all the conflicting classes to be unassigned), pick one of the suggestions, or start resolving the conflicts manually by selecting a new placement for one of the conflicting classes. This process can continue until all conflicts are resolved manually or a suggestion resolving all the remaining conflicts is found.

### 6.3 Data Consistency

Often during the early stages of the timetabling process, the input data provided by schedule managers are inconsistent. This means that the problem is over-constrained, without any complete feasible solution. A very important aspect of the timetabling system is therefore an ability to provide enough information back to the timetablers describing why the solver is not able to find a complete solution.



In prior work on this problem [15], a learning technique, called conflict-based statistics, was developed that helps the solver to escape from a local optimum. This helps to avoid repetitive, unsuitable assignments to a class. In particular, conflicts caused by a past assignment, along with the assignment that caused them, are stored in memory. This learned information gathered during the search is also highly useful in providing the user with relevant data about inconsistencies and for highlighting difficult situations occurring in the problem.

## 7 Implemented System

The system being implemented at Purdue University has been designed as a multi-user application with a completely web-based interface. The primary technologies used are the Enterprise Edition of Java 2 (J2EE), Hibernate, and an Oracle database.

From the initial planning stages, it was recognized that there were a wide variety of timetabling needs in different University departments and a wide range in departmental schedule manager's comfort level with automating the timetabling process. The application has, therefore, been conceived of as a flexible tool to help departmental timetablers with the process rather than a completely automatic system without human interaction.

At the time of this writing, the automated timetabling system has been used to create the large lecture timetable for the past three terms. It has also been used by the computing lab manager and seven departmental or college schedule managers to solve a range of different problems for the past term. The system is planned to be implemented campus-wide in January 2007 for developing the Fall timetable. Prior to full-scale use, considerable training is planned for schedule managers. Individuals with experience manually creating timetables are not necessarily used to formulating the rules they use as a set of constraints. Some schedule managers seem apprehensive about spelling out, even for themselves, the actual decision rules and priorities they apply when constructing a timetable. This may be because many decisions are based on political rather than objective criteria. For the initial campus-wide implementation, therefore, users who are more comfortable with their manually developed timetables will only be required to use the system for entry of class data and checking for inconsistencies in their solution. Departments that want to use all or most of the features of the system will be trained to use the solver on the constraints they enter. It is anticipated that the departmental schedule managers will want to use more and more capabilities as they become comfortable with the system and experienced with using the solver to help place additional classes.

### 7.1 Spring 2007 Timetables

Table 4 shows a summary of solutions for the individual problems that were discussed in Section 2.2 i.e., the large lecture problem (LLR), the centrally timetabled computer laboratory problem (LAB), and two different departmental problems (D1, D2). The column titled *Final* contains the results that were

**Table 4.** Individual solution properties of four problems

LLR (804 variables)	Final	Run separately	Run combined
Time [min]	–	$5.2 \pm 4.9$	–
Student conflicts	1207	$756.8 \pm 25.1$	$723.6 \pm 28.0$
Preferred time [%]	84.7	$89.9 \pm 0.7$	$89.7 \pm 1.7$
Preferred room [%]	88.9	$91.9 \pm 0.9$	$92.0 \pm 0.8$
Preferred distribution [%]	66.7	$87.5 \pm 5.9$	$70.0 \pm 6.0$
D1 (440 variables)	Final	Run separately	Run combined
Time [min]	–	$20.8 \pm 3.6$	–
Student conflicts	11	$12.3 \pm 2.3$	$13.2 \pm 4.1$
Preferred time [%]	67.4	$78.8 \pm 2.0$	$81.1 \pm 1.4$
Preferred room [%]	76.2	$78.2 \pm 2.2$	$77.1 \pm 1.6$
Preferred distribution [%]	57.1	$61.9 \pm 4.1$	$67.4 \pm 4.5$
D2 (69 variables)	Final	Run separately	Run combined
Time [min]	–	$0.08 \pm 0.07$	–
Student conflicts	3	$0.6 \pm 1.0$	$3.4 \pm 1.7$
Preferred time [%]	81.6	$95.9 \pm 1.0$	$95.9 \pm 1.7$
Preferred room [%]	100.0	$100.0 \pm 0.0$	$100.0 \pm 0.0$
Preferred distribution [%]	100.0	$100.0 \pm 0.0$	$100.0 \pm 0.0$
LAB (443 variables)	Final	Run separately	Run combined
Time [min]	–	$5.3 \pm 3.4$	–
Student conflicts	14	$14.0 \pm 0.0$	$14.2 \pm 0.4$
Preferred time [%]	87.6	$94.5 \pm 1.4$	$96.6 \pm 0.8$
Preferred room [%]	75.8	$78.8 \pm 0.5$	$78.6 \pm 0.8$
Preferred distribution [%]	68.0	$75.5 \pm 4.2$	$78.0 \pm 3.4$

produced by schedule deputies using the timetabling application for Spring 2007. These solutions were initially computed individually using the automated solver; however, changes were also made using the interactive solver. The column *Run separately* contains results from 10 individual test runs. The LLR problem was solved first. Problems D1 and D2 were solved on top of the LLR solution. The LAB problem was solved at the end, on top of the LLR, D1 and D2 solutions that were produced in the same test run. In each row, the average and RMS (root-mean-square) variances of the best solutions found within a 30 minute time limit are displayed. The column *Run combined* contains average results from 10 combined test runs. In these tests, all four problems were solved together within a 120 minute long time frame.

During work on the Spring 2007 data set, the solver was able to provide consistent solutions of high quality. The difference in properties between solutions to individual problems are caused primarily by differences in the characteristics of these problems. For example, there are 27,881 students involved in the LLR problem, with each student taking 3.15 LLR classes on average, but there are

only 8,408 students in LAB problem, with each students taking only 1.14 LAB classes. Section 2.2 discusses these differences in more detail. The input data for each department has also been entered by a different schedule manager for each problem and there are sizable differences in the number and quality of preferences/requirements entered by each manager. This leaves the solver with a much different capacity for optimization in each problem. In many cases, the managers seem to be trying to convince the solver to mimic the properties of the manually made solutions they were accustomed to. In particular, the large increase in student conflicts between the computed best solution and the final solution committed by the schedule manager in the large lecture problem is largely attributable to adjustments to accommodate faculty time preferences. These had been the primary criteria in manually building timetables since data on student conflicts were not available to be considered in the past.

## 7.2 Data Sets

Input data sets for the timetabling problems discussed above are available in an easily readable XML format at <http://www.smas.purdue.edu/research>. In order to comply with data security policies, these data sets have been purged of all private information; however, they do retain all of the complexity of the Purdue University timetabling problem that has been encountered so far. Future expansion of these pages will include new timetabling input data sets as well as additional information about ongoing research. A verification mechanism may also be developed for solutions to the timetabling problems that have been included, as well as an open-source version of the timetabling solver and/or the entire timetabling application. It is hoped that the format in which the data are presented on this page will create a foundation for a widely acceptable format for interchange of complex university course timetabling benchmarks.

## 8 Conclusions

Based on the results of this project, it is clear that complex university course timetabling problems can be solved at a level where these solutions are of practical use in the real world. Creating systems to do so is still not an easy process, but it is possible to develop effective solutions using methods that are readily available. The biggest challenges at this point appear to be understanding the structures in the problem being considered and addressing the concerns that users have with timetabling beyond the basic solution method.

Considering the problem in a layered framework (constraint satisfaction, course timetabling, presentation) was of significant help in developing a flexible enough approach to solving the problem that it could withstand the numerous adaptations necessary to extend the solution capabilities to cover the university-wide problem. This type of framework may also be useful for considering other types of problems. The examination timetabling problem, for example, would fit nicely on top of the solver used here with a new data model in the presentation layer and some adaptation to the timetabling model.

Creating a data model that can simplify the complex course structures encountered into a more manageable series of classes is also very important. While it is still likely that institutions with significantly different structures for their courses will require different data models, and possibly some difference in their timetabling models, it is clear from the work that has been done here that with sufficient effort it is possible to develop models comprehensive enough to be used on a wide set of problems. Aside from some interfaces for importing data from other systems, the timetabling application developed as a result of this project should be usable at a large number of other universities with similar complexity in course structures.

*Acknowledgements.* Hana Rudová's work is supported by the Ministry of Education, Youth and Sports of the Czech Republic, research intent No. 0021622419.

## References

1. Abdennadher, S., Marte, M.: University course timetabling using constraint handling rules. *Journal of Applied Artificial Intelligence* 14, 311–326 (2000)
2. Amintoosi, M., Haddadnia, J.: Feature selection in a fuzzy student sectioning algorithm. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 147–160. Springer, Heidelberg (2005)
3. Aubin, J., Ferland, J.A.: A large scale timetabling problem. *Computers and Operations Research* 16, 67–77 (1989)
4. Barták, R., Müller, T., Hana Rudová, H.: A new approach to modeling and solving minimal perturbation problems. In: Apt, K.R., Fages, F., Rossi, F., Szeredi, P., Vánca, J. (eds.) *CSCLP 2003*. LNCS (LNAI), vol. 3010, pp. 233–249. Springer, Heidelberg (2004)
5. Cambazard, H., Demazeau, F., Jussien, N., David, P.: Interactively solving school timetabling problems using extensions of constraint programming. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 190–207. Springer, Heidelberg (2005)
6. Carter, M.W.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 64–82. Springer, Heidelberg (2001)
7. Carter, M.W., Gilbert Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
8. Dechter, R.: *Constraint Processing*. Morgan Kaufmann, San Mateo, CA (2003)
9. Guéret, C., Jussien, N., Boizumault, P., Prins, C.: Building university timetables using constraint logic programming. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 130–145. Springer, Heidelberg (1996)
10. Hertz, A.: Tabu search for large scale timetabling problems. *European Journal of Operational Research* 54, 39–47 (1991)
11. McCollum, B.: A perspective on bridging the gap in university timetabling. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006*. LNCS, vol. 3867, pp. 3–23. Springer, Heidelberg (2007)
12. Mooney, E.L., Rardin, R.L., Parmenter, W.J.: Large scale classroom scheduling. *IIE Transactions* 28, 369–378 (1996)

13. Müller, T.: Constraint-based Timetabling. Ph.D. Thesis, Charles University in Prague, Faculty of Mathematics and Physics (2005)
14. Müller, T., Barták, R., Rudová, H.: Conflict-based statistics. In: Gottlieb, J., Landa Silva, D., Musliu, N., Soubeiga, E. (eds.): EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics, University of Nottingham (2004)
15. Müller, T., Barták, R., Rudová, H.: Minimal perturbation problem in course timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 126–146. Springer, Heidelberg (2005)
16. Petrovic, S., Burke, E.K.: University timetabling. In: Leung, J.Y.-T. (ed.) The Handbook of Scheduling: Algorithms, Models, and Performance Analysis, ch. 45, CRC Press, Boca Raton, FL (2004)
17. Qualizza, A., Serafini, P.: A column generation scheme for faculty timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 161–173. Springer, Heidelberg (2005)
18. Rudová, H., Murray, K.: University course timetabling with soft constraints. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 310–328. Springer, Heidelberg (2003)
19. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)

# Timetabling Problems at the TU Eindhoven

John van den Broek, Cor Hurkens, and Gerhard Woeginger

Department of Mathematics and Computer Science,  
Eindhoven University of Technology,  
Den Dolech 2, 5600 MB Eindhoven, The Netherlands  
j.j.j.v.d.broek@tue.nl, {wscor,gwoegi}@win.tue.nl

**Abstract.** The students of the Industrial Design department at the TU Eindhoven are allowed to design part of their curriculum by selecting courses from a huge course pool. They do this by handing in ordered preference lists with their favorite courses for the forthcoming time period. Based on this information (and on many other constraints), the department then assigns courses to students. Until recently, the assignment was computed by human schedulers who used a quite straightforward greedy approach. In 2005, however, the number of students increased substantially, and as a consequence the greedy approach no longer yielded acceptable results.

This paper discusses the solution of this real-world timetabling problem. We present a complete mathematical formulation of it, and we explain all the constraints resulting from the situation in Eindhoven. We solve the problem using lexicographical optimization with four subproblems. For all four subproblems, an elegant integer linear programming model is given which easily can be put into CPLEX. Finally, we report on our computational experiments and results around the Eindhoven real-world data.

## 1 Introduction

In February 2005, outraged students of the Industrial Design department were protesting at the TU Eindhoven (The Netherlands). Uproar and revolt were in the air. What had happened? Here is the story. The academic year of the roughly 350 students of Industrial Design is split into a number of periods. In every period, every student must do a number of small *courses*. There is a pool of roughly 55 courses to choose from, and every student submits an ordered preference list with his/her 10 favorite courses to the department. Based on all the ordered preference lists, a *scheduler* at the department then assigns roughly four courses to every student. Until recently, the scheduler was a human decision-maker who essentially applied a hand-woven greedy assignment procedure.

In February 2005, the students were absolutely dissatisfied with the work of the human scheduler: many of them did not get the courses which they would have liked to get; many of them were assigned to courses which they really did not want to do; and more than 150 out of the 350 students received courses that were not listed on their preference list!

The department of Industrial Design realized that they had a problem. They also realized that they did not know how to solve this problem. The number of students had increased substantially, and the timetabling problem had become much larger, much harder, and much more complex. Hence, the department contacted the local experts on the campus: us. They were hoping to find a somewhat better assignment through computer programs. They explained their problem to us (in a certain problem formulation No. 1), and we happily told them that we would be able to solve it: the problem (in formulation No. 1) could be modeled as a network flow problem, and hence is solvable in polynomial time. Unfortunately, it turned out that formulation No. 1 was not a complete formulation of the problem: they had forgotten to inform us about a number of additional restrictions that lead to a new, more difficult assignment problem (in formulation No. 2), which eventually turned out to be NP-hard.

This paper is a report on the course assignment problem of the Industrial Design department. We describe the assignment problem in its (incomplete) formulation No. 1 and in its (complete) formulation No. 2. We show that formulation No. 1 yields a tractable problem, whereas formulation No. 2 yields an intractable problem. Our main contribution is a careful case study of the complete problem formulation. We design an elegant integer linear programming model for it, with roughly 9000 variables and roughly 7000 constraints. Putting this ILP model into CPLEX yields excellent results within moderate computation times. We describe the ILP model in detail, and we report on our computational experiments with the real-world data of the Industrial Design department.

*Structure of the paper.* The rest of the paper is structured in the following way. In Section 2 we give a literature review of university and school timetabling. Section 3 contains a detailed description of the problem we solved for the department of Industrial Design. The problem is formulated as an integer linear program which is described in Section 4. Section 5 contains the computational results. Some conclusions are given in Section 6.

## 2 Literature Review

The literature contains a large number of variants of the timetabling problem. These variants differ from each other by the type of institution involved (university or high school) and by the type of constraints. The annotated bibliography of timetable construction by Schmidt and Ströhlein [22] lists many papers that appeared before 1980. Schaerf [21] gives a survey of the various formulations of timetabling problems and classifies the timetabling problem into the following three main classes: school timetabling, examination timetabling and course timetabling. Of course this classification is crude, and there are many real-world timetabling problems that fall in between two of these classes. For surveys of timetabling methods and applications see de Werra [11], Burke et al. [3], Carter and Laporte [8] and Burke and Petrovic [5].

The basic school timetabling problem is also known as the *class–teacher model*. The simplest problem consists in assigning lectures to periods in such a way that no teacher or class is involved in more than one lecture at a time. Even et al. [13] proved that there always exists a solution of this simplest version of the school timetabling problem, unless a teacher or class is involved in more lectures than there are time slots. Alternative formulations of the school timetabling problem with more constraints can be found for example in Even et al. [13], Garey and Johnson [15] and de Werra [11]. Daskalaki and Birbas [10] provide an integer programming formulation of the class–teacher problem and solve it with a two-stage relaxation procedure. The problem is formulated as a set packing problem with side constraints by Avella and Vasilev [2]. They tighten the formulation by adding the valid inequalities of the Set Packing polytope and introduce some new valid inequalities.

University timetabling can be classified into two categories: course and examination timetabling. Petrovic and Burke [19] discuss problem statements and give an overview on recent research results on university timetabling. The main differences between course timetabling and examination timetabling are that examination timetabling has only one exam for each course, that the time conflict condition is strict, and that several exams can be done simultaneously in one room. Examples for additional soft constraints are: students can do at most one exam per day, and students may not have too many consecutive exams. Schaerf [21] gives an integer linear programming formulation of the examination timetabling problem and describes some alternative variants of the problem. Carter and Laporte [7] provide an overview on examination timetabling.

The course timetabling problem consists in scheduling a set of lectures for each course within a given number of rooms and time period. The main difference from the school timetabling problem is that university courses can have common students, whereas school classes are disjoint sets of students. De Werra [11] gives a binary integer programming formulation. An overview on course timetabling problems is given by Carter and Laporte [8] and Schaerf [21] discusses some of the most common variants of the basic formulation. The design and implementation of a decision support system for constructing a combined university course-examination timetable is reported by Dimopoulou and Miliotis [12]. They also take into account the increased flexibility of students' preferences for specific classes.

One variant is called the *grouping subproblem* or *student scheduling problem*. If the number of students is too large for one room, courses are split into groups of students and there are conditions on the minimum and maximum number of students that can be assigned to each group. A student is required to take a certain number of courses, which they have to select themselves after a timetable is made available. The problem consists of assigning a student to a specific group of a course for a given fixed timetable such that students are satisfied and there are no time conflicts, see Busam [6], Feldman and Golubic [14] and Laporte and Desrochers [16].



Cheng et al. [9] discuss the Student Scheduling Problem (SSP) as it generally applies to high schools in North America. They define the problem as the assignment of courses and a specific section to each student. The objective is to fulfill student requests, providing a conflict-free schedule. They show that the problem is NP-hard and discuss various multi-commodity flow formulations with fractional capacities and integral gains. The main difference between the SSP and our timetabling problem is that for the SSP all courses on the preference list of the students have to be assigned to students. This results with most practical cases in an empty feasible solution set.

Laporte and Desrochers [16] give a mathematical formulation of the student scheduling problem. They formulate the problem as an optimization problem splitting the requirements into hard and soft ones. The only hard constraint in their model is that student course selections must be respected. Time conflicts for students are soft constraints. When time conflicts occur students are advised to make a different course selection. The problem is then solved in three phases: in the first one the algorithm searches for an admissible solution, in the second section enrollments are balanced and in the third the room capacities have to be respected. Tripathy [23] formulated the student scheduling problem as an integer linear programming problem and uses Lagrangian Relaxation to solve it. Sabin and Winter [20] use a greedy approach that is moderated by an intelligent ordering of the students. Miyaji et al. [18] apply goal programming.

McCollum [17] explains that for university timetabling there is still a gap between a successful research project and what is needed in practice. He tries to bridge this gap between research and practice by providing up-to-date information from practice which is needed by researchers. Burke et al. [4] and Zampieri and Schaefer [24] note that many of the search methodologies described in the literature are not applicable in most educational institutions, because they are simplified too much.

Carter and Laporte [8] note that they were ‘somewhat surprised to discover that there are very few course timetabling papers that actually report that the (research) methods have been implemented and used in institution’. McCollum [17] explains that the situation has hardly changed in the last decade. Our paper is an example of a successful implementation of a mathematical programming model for a specific course timetabling problem.

### 3 Problem Description

At our first meeting, the Industrial Design department explained the problem to us in a certain problem formulation No. 1; see Section 3.1. This problem can be modeled as a network flow problem, and hence is solvable in polynomial time; see Ahuja et al. [1].

Unfortunately, we learnt after some time that formulation No. 1 was *not* a complete formulation of the problem. They actually had forgotten to tell us about a number of additional restrictions that lead us to a new, more difficult assignment problem formulation No. 2. Section 3.2 describes formulation No. 2.

**Table 1.** Example of preference lists

Student	$r_s$	P1	P2	P3	...	P10
s040202	4	DAC03	DA247	DA125	...	DA405
s040203	4	DA619	DA125	DA201	...	DA616
s040204	4	DA418	DA242	DA402	...	DA621

### 3.1 Problem Formulation No. 1

At the first meeting with the Industrial Design department, we were told that every student hands in a preference list of at most 10 courses and requests a certain number of courses. The only constraints are that a student cannot do two courses at the same time and there is a maximum number of students that can be assigned to a course. This section contains a more detailed description of problem formulation No. 1.

A set  $C$  of courses and for each course  $c$  an *upper bound*  $C_c^{\max}$  on the number of students is given. This number depends on the preference of the teacher and the room capacity in which the course is given. Each course has one weekly meeting time which is already fixed. This weekly meeting time always consists of two consecutive hours. Two such consecutive hours are defined as one *time slot*. The weekly meeting time of a course is chosen from a set  $T$  of disjoint time slots.  $T(c)$  is defined as the time slot which is the weekly meeting time of course  $c$ . Hence, one of the constraints in the model is that courses  $c_i$  and  $c_j$  cannot be assigned to one student if  $T(c_i) = T(c_j)$ .

We define  $S$  as the set of students. For each student  $s$  the requested number  $r_s$  of courses is given.  $P_s$  is defined as the set of positions on the preference list for which student  $s$  filled in a course. Most students have  $P_s = \{1, \dots, 10\}$ . There are also students that hand in a smaller preference list. For instance, a student almost finishing his bachelor degree and with only one course left to do, which has to be a math course, hands in a preference list with only math courses. For a student  $s$  with only six courses on its preference list we have  $P_s = \{1, \dots, 6\}$ . Table 1 gives a few examples of preference lists. Column  $P_i$  gives the encoded course name of the course on position  $i$  of the preference list. The parameter  $c_{sp}$  is introduced and is equal to  $c$  if course  $c$  is on position  $p$  of the preference list of student  $s$ .

*In summary*, the input of problem formulation No. 1 consists of

- a set  $T$  of time slots;
- a set  $C$  of courses; for every course  $c \in C$  a time slot  $T(c)$  and a maximum number  $C_c^{\max}$  of participating students is given;
- a set  $S$  of students; for every student  $s \in S$  a set  $P_s$  of filled positions of the preference list, a course  $c_{sp}$  for each position  $p \in P_s$  and a requested number  $r_s$  of courses is given.

The goal is to assign as many courses to students as possible, while

- the number of courses assigned to student  $s$  does not exceed the requested number  $r_s$ ,
- courses assigned to a student are on his preference list,
- courses assigned to a student do not conflict in time,
- no course exceeds its maximum number of assigned students.

This problem can be modeled as a network flow problem. A description of this network flow model is given in Appendix [A](#).

### 3.2 Problem Formulation No. 2

When we received the first data set from the Industrial Design department, we were very surprised: there suddenly were also *lower bounds*  $C_c^{\min}$  on the number of students participating in course  $c$ . This yields the new constraint that a course either will not be given at all, or otherwise has at least  $C_c^{\min}$  participating students. This new constraint cannot be modeled as a flow-constraint, and hence the maximum flow model in Appendix [A](#) becomes obsolete. In fact, the new constraint makes the problem NP-hard; see Appendix [B](#). After looking at the data more carefully and after conversations with the Industrial Design department we noticed there were a lot more restrictions. This section explains these extra restrictions and defines the problem in more detail.

An academic year is divided into a certain number of teaching periods. For instance, the academic year 2005–06 is divided into six teaching periods. We define such a teaching period as a *block*. The Industrial Design department wants us to schedule two consecutive blocks simultaneously. Therefore, set  $B$  is introduced as the set of blocks that have to be scheduled simultaneously.

In problem formulation No. 1 we assumed the *workload* of all courses was equal. However, there are courses with a workload of 40 hours and courses with a workload of 80 hours. In the remainder of this paper a workload of 1 corresponds with a workload of 40 hours. In Appendix B we prove that having courses with a workload 1 and courses with a workload 2 makes the problem NP-hard. For each course  $c \in C$  and block  $b \in B$  the parameter  $w(c, b)$  is defined as the workload of course  $c$  in block  $b$ . Hence for a course  $c$  with a workload of 80 hours in block  $b$  we have  $w(c, b) = 2$ .

In problem formulation No. 2 the definition of  $r_s$  is adjusted into the requested workload of student  $s$  for  $|B|$  blocks together. For every student  $s$ , a maximum requested workload  $r_{sb}$  for each block  $b \in B$  is given separately, because the requested workload of a student is not always equally divided over all blocks  $b \in B$ . For instance, if student  $s$  has to do a practical training in block  $b_2$  he has  $r_s = 2$ ,  $r_{sb_1} = 2$  and  $r_{sb_2} = 0$ .

It was assumed in problem formulation No. 1 that a course has one meeting every week, hence it has one time slot. But there are also courses which have two weekly meetings, and hence two time slots. If courses with two time slots are introduced into problem formulation No. 1, the problem cannot be modeled as a network flow problem.

**Table 2.** Examples of courses

Course	Section	Time slots of meetings	wlb1	wlb2	Min	Max
DA242	DAG242-1	B1TM2, B1TA1	1	0	0	30
	DAG242-2	B1TM2, B1TA2	1	0	0	30
	DAG242-3	B1TM2, B1WA1	1	0	0	30
	DAG242-4	B1TM2, B1WA1	1	0	0	30
	DAG242-5	B1TM2, B1WA2	1	0	0	30
DA247	DAG247-1	B1WA2, B2WA2	1	1	5	15
	DAG247-2	B1WA2, B2WA2	1	1	5	15

The set  $C$  of courses contains courses with multiple sections, meaning that the course is repeated during the week. Table 2 contains course DA242 as an example. For example, time slot B1TM2 stands for the second part of Tuesday morning in block 1. The workloads of a course in block 1 and 2 are denoted by  $wlb1$  and  $wlb2$ . The course DA242 has five sections which all have two time slots.

We define  $I$  as the set of sections offered to the students. For every section  $i \in I$  its course  $c(i) \in C$  is given, a minimum number  $C_i^{\min}$  and a maximum number  $C_i^{\max}$  of students. The meeting times for each section  $i \in I$  are given as the set of time slots  $T(i) \subseteq T$ . There are a few courses, for example literature studies, which are not assigned to a time slot and thus  $T(i) = \emptyset$ .

Another constraint arises if students have specific needs, for instance when they almost finish their studies and only have one course left to pass. Then a course on the preference list of the student can be set to *urgent*. As long as the maximum number of students (all with an urgency) is not assigned to this course, the course has to be assigned to the student. A course which is urgent for one student has to be given. In this case, it doesn't matter whether the minimum number of students is reached or not. We define  $U$  as the set containing all combinations  $(s, p)$  for which course  $c_{sp}$  is urgent for student  $s$ .

A few courses have meeting times which are spread over two blocks. See for example course DA247 in Table 2. This course has two sections and a total workload of two which is equally spread over the two blocks. If a student is assigned to a section of this course in one block he needs to be assigned to the same section of this course in the next block. Hence, it is also possible that courses are given in two blocks which are not scheduled simultaneously. If this occurs, this implies there are students already preassigned to sections if the schedule of the second block is made. Therefore, we introduce the set  $F$  of *fixations* which contains combinations  $(s, p, i)$  for which section  $i$  of course  $c_{sp}$  is already assigned to student  $s$ . This results in hard constraints that do not lead to an infeasible solution, because these students are assigned in the scheduling period before.

*In summary*, the input of problem formulation No. 2 consists of

- a set  $B$  of blocks that have to be scheduled simultaneously;
- a set  $T$  of time slots;
- a set  $C$  of courses; for every course  $c$  its workload  $w(c, b)$  for each block  $b$  is given;

- a set  $S$  of students; for every student  $s$  a total requested workload  $r_s$ , a requested workload  $r_{sb}$  for each block separately, a set  $P_s$  of filled positions on the preference list and for each position  $p \in P_s$  a course  $c_{sp}$  is given;
- a set  $I$  of sections; for every section  $i$  its course  $c(i)$ , a minimum  $C_i^{\min}$  and maximum  $C_i^{\max}$  number of students and a set of time slots  $T(i) \subseteq T$  is given;
- a set  $U$  of combinations  $(s, p)$  for which course  $c_{sp}$  is urgent for student  $s$ ;
- a set  $F$  of combinations  $(s, p, i)$  for which section  $i$  of course  $c_{sp}$  is already preassigned to student  $s$ .

Our main goal is to assign workload to students as much as possible, while

- maintaining the number of students in a section below a maximum size prescribed,
- the total workload assigned to student  $s$  is less than or equal to  $r_s$ ,
- the workload assigned to student  $s$  in block  $b$  is less than or equal to  $r_{sb}$ ,
- sections assigned to a student do not conflict in time,
- students are only assigned to a section of a course on their preference list,
- students are only assigned to one section of a course,
- student  $s$  is assigned to section  $i$  if  $(s, p, i) \in F$ .

Soft constraints are, for example, spreading students over sections, a section needing to be assigned to at least a certain minimum number of students and student  $s$  having to be assigned to course  $c_{sp}$  if  $(s, p) \in U$ .

## 4 The Integer Linear Programming Model

The problem is formulated and solved as a *lexicographic optimization* problem. Lexicographic optimization is a form of multi-criteria optimization in which the various objectives  $f_i, i = 1, \dots, m$  cannot be quantitatively traded off between each other. If a solution  $x$  minimizes  $f_1$ , then a solution  $x'$  minimizes  $f_2$  if the condition  $f_1(x) = f_1(x')$  is satisfied. In general,  $x^*$  minimizes  $f_i$  under the constraining conditions that  $f_1(x^*) = f_1(x_1), \dots, f_{i-1}(x^*) = f_{i-1}(x_{i-1})$  where  $x_j$  minimizes  $f_1, \dots, f_j$  for  $1 \leq j \leq i - 1$ .

The timetabling problem is split into four subproblems which are formulated as an integer linear programming problem. The goals of the four subproblems are:

1. Maximize the number of assigned courses with an urgency.
2. Minimize the shortage of students to reach the minimum number of students of a section. Because of urgencies, some sections must be taught, but do not have enough students with this course on their preference list. We assign as many students as possible to those sections.
3. Maximize the total assigned workload. We try to assign a workload  $r_s$  to every student  $s$ .
4. ‘Optimize’ the timetable. For example by assigning courses to students which rank high on their preference list.

All parameters are already introduced in Section 3. Left to define are the decision variables. These are defined as follows:

$$x_{sp} := \begin{cases} 1 & \text{if course } c_{sp} \text{ is assigned to student } s \\ 0 & \text{otherwise} \end{cases}$$

$$y_i := \begin{cases} 1 & \text{if section } i \text{ is assigned to one or more students} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{spi} := \begin{cases} 1 & \text{if section } i \text{ of course } c_{sp} \text{ is assigned to student } s \\ 0 & \text{otherwise.} \end{cases}$$

The following constraints have to be fulfilled in all four subproblems:

$$\sum_{i \in I | c_{sp}=c(i)} z_{spi} = x_{sp} \quad \forall s \in S, \forall p \in P_s \quad (1)$$

$$\sum_{p \in P_s} \sum_{i \in I | c_{sp}=c(i)} w(c_{sp}, b) z_{spi} \leq r_{sb} \quad \forall s \in S, \forall b \in B \quad (2)$$

$$\sum_{p \in P_s} \sum_{i \in I | c_{sp}=c(i)} \sum_{b \in B} w(c_{sp}, b) z_{spi} \leq r_s \quad \forall s \in S \quad (3)$$

$$\sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} \leq C_i^{\max} y_i \quad \forall i \in I \quad (4)$$

$$\sum_{p \in P_s} \sum_{i \in I | c_{sp}=c(i), t \in T(i)} z_{spi} \leq 1 \quad \forall s \in S, \forall t \in T \quad (5)$$

$$z_{spi} = 1 \quad \forall s \in S, \forall p \in P_s, \forall i \in I | (s, p, i) \in F \quad (6)$$

$$x_{sp} \in \{0, 1\} \quad \forall s \in S, \forall p \in P_s \quad (7)$$

$$y_i \in \{0, 1\} \quad \forall i \in I \quad (8)$$

$$z_{spi} \in \{0, 1\} \quad \forall s \in S, \forall p \in P_s, \forall i \in I. \quad (9)$$

Constraint (1) ensures that at most one section of a course is assigned to a student. The workload assigned to a student has to be less than or equal to the requested workload of each block separately and all blocks together. This is fulfilled by constraints (2) and (3). Constraint (4) enforces that the maximum number of students for a section is not exceeded and constraint (5) ensures that at each time slot only one section is assigned to each student. If  $(s, p, i) \in F$  then section  $i$  of course  $c_{sp}$  has to be assigned to student  $s$ , which is fulfilled by constraint (6).

The goal of the first subproblem is to maximize the number of assigned courses with an urgency. The constraint that a section needs to have more than a minimum number of students is not a restriction in this subproblem, because at least

one section of a course must be given if there is a student with an urgency for this course. This first subproblem can be solved with the following ILP formulation:

$$U^{\max} = \max \sum_{(s,p) \in U} x_{sp}$$

$$(x, y, z) \text{ satisfy (1)–(9).}$$

The next step is to minimize the shortage of students to reach the minimum number of students of a section, keeping the maximum number of assigned courses with an urgency equal to  $U^{\max}$ . There are sections that have to be given because they are assigned to students with an urgency for the corresponding course. Those sections are assigned to other students such that the minimum number of students for those sections is reached. The decision variable  $s_i$  is defined as the shortage of students for section  $i$ . This variable gets a value larger than zero if it is not possible to assign section  $i$  to the minimum number  $C_i^{\min}$  of students. The second subproblem minimizes the total shortage  $S^{\min}$  of students. This results into the following ILP formulation:

$$\min \sum_{i \in I} s_i = S^{\min}$$

$$\sum_{(s,p) \in U} x_{sp} = U^{\max}$$

$$\sum_{s \in S} \sum_{p \in P_s, c_{sp} = c(i)} z_{spi} + s_i \geq C_i^{\min} y_i \quad \forall i \in I$$

$$s_i \in \mathbb{Z}_+, \quad \forall i \in I$$

$$(x, y, z) \text{ satisfy (1)–(9).}$$

The third subproblem maximizes the total workload assigned to students with the restrictions that  $U^{\max}$  and  $S^{\min}$  keep their optimal values. This maximum workload is denoted by  $W^{\max}$  and is determined by the following model:

$$\max \sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b) x_{sp} = W^{\max}$$

$$\sum_{i \in I} s_i = S^{\min}$$

$$\sum_{(s,p) \in U} x_{sp} = U^{\max}$$

$$\sum_{s \in S} \sum_{p \in P_s, c_{sp} = c(i)} z_{spi} + s_i \geq C_i^{\min} y_i, \quad \forall i \in I$$

$$s_i \in \mathbb{Z}_+, \quad \forall i \in I$$

$$(x, y, z) \text{ satisfy (1)–(9).}$$

To ‘optimize’ the final timetable we assign courses as high as possible on the preference lists, spread the students as equally as possible over the sections of a course and discourage the possibility that one student gets a lot of courses which are on the bottom of his preference list. Therefore, the fourth subproblem is solved. The objective function is separated into three terms and has to be minimized under the restrictions that  $U^{\max}$ ,  $S^{\min}$  and  $W^{\max}$  keep their optimal values.

The term in the objective function to assign courses as high as possible on the preference lists is  $W_p \sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b)(82 - (10 - p)^2)x_{sp}$ . Assigning a course on top of a preference list,  $p = 1$  for this course, adds a lot less to the objective function than assigning a course on the bottom of the list,  $p = 10$  for this course.  $W_p$  is a weighting factor and also the workload is taken into account.

If a course has multiple sections, students have to be spread as equally as possible over the sections. Therefore,  $I_c^{\max}$  is introduced as the number of students assigned to the section of course  $c$  with the most students assigned. Also the spread  $S_c$  of course  $c$  is introduced and is equal to the sum over all sections of the difference between  $I_c^{\max}$  and the assigned number of students in each section.  $S_c$  is added to the objective function with a weighting factor  $W_s$ .

We also discourage the possibility that one student gets a lot of courses from the 7th to 10th position of his preference list. A constraint is added to the model that checks whether a student gets more than one course from these positions. If so, then a penalty  $W_e$  is paid for each ‘extra’ course from these positions. Therefore, the decision variable  $E_s$  is introduced for every student  $s$ . This variable is equal to the ‘extra’ number of courses assigned to student  $s$  which are from the 7th to 10th position of his preference list.

This results in the final ILP formulation:

$$\begin{aligned} \min W_p \sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b)(82 - (10 - p)^2)x_{sp} &+ W_s \sum_{c \in C} S_c &+ W_e \sum_{s \in S} E_s \\ \sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} &\leq I_{c(i)}^{\max} &\forall i \in I \\ \sum_{i \in I | c=c(i)} \left( I_c^{\max} - \sum_{s \in S} \sum_{p \in P_s, c_{sp}=c} z_{spi} \right) &= S_c &\forall c \in C \\ \sum_{p=7}^{10} x_{sp} &\leq 1 + E_s &\forall s \in S \end{aligned}$$

$$\sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b)x_{sp} = W^{\max}$$

$$\sum_{i \in I} s_i = S^{\min}$$



**Table 3.** Input information for academic year 2005–06

Blocks	$ S $	$ C $	$ I $	$ U $	Offered wl	Requested wl
1 & 2	356	51	79	590	1504	1416
3 & 4	328	64	88	279	1545	1288
5 & 6	302	58	89	151	1544	1333

$$\sum_{(s,p) \in U} x_{sp} = U^{\max}$$

$$\sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} + s_i \geq C_i^{\min} y_i, \quad \forall i \in I$$

$$E_s \in \mathbb{Z}_+ \quad \forall s \in S$$

$$I_c^{\max}, S_c \in \mathbb{Z}_+ \quad \forall c \in C$$

$$s_i \in \mathbb{Z}_+ \quad \forall i \in I$$

$(x, y, z)$  satisfy (1)–(9).

## 5 The Computational Results

The computational results for the academic year 2005–2006 are given in this section. This academic year was divided into six blocks. Blocks 1 & 2, blocks 3 & 4 and blocks 5 & 6 were scheduled simultaneously.

In all blocks the meetings were on Tuesday morning, Tuesday afternoon, Wednesday morning and Wednesday afternoon. Every morning and afternoon was split into two parts. So both blocks contained eight time slots. More details about the input are given in Table 3. The abbreviation wl stands for workload.

The number of students that requested workload in blocks 1 & 2 was 356 and the total workload they requested was 1416. Hence, for each block, an average of two courses of the preference list of 10 courses have to be assigned. The number of students requesting workload decreased during the academic year, because of students who are doing a practical training and students who are finishing their studies. In blocks 5 & 6 the average requested workload per student is larger than in blocks 3 & 4, which is caused by students who still hope to reach the required workload for the academic year by doing some ‘extra’ courses.

Note that the large number of urgencies in blocks 1 & 2 can be explained by the fact that first year students are preassigned to courses, because they are not able to make a choice themselves. All first year students have six compulsory courses, which they have to do in the first year. These six courses are set as urgent and only four out of these six courses can be done in each block. The urgencies in blocks 5 & 6 can be explained by students who are finishing their education and only have some specific courses left to do.

The models introduced in Section 4 are solved by the standard IP solver CPLEX 10.0. The computations are done on an Intel Pentium M, 2.0 GHz

**Table 4.** Results for the academic year 2005–06

	Block 1 & 2	Block 3 & 4	Block 5 & 6
Runtime CPLEX (s)	1.38	1.53	1.67
$U^{\max}$	439	273	134
$S^{\min}$	0	0	0
$W^{\max}$	1369	1261	1300
Average position	3.30	3.64	3.87
Bad positions	8	16	39

processor with 1.0 GB internal memory. The values of the weighting factors were  $W_p = 10$ ,  $W_s = 1$  and  $W_e = 100$ . The results for the academic year 2005–06 are given in Table 4.

The computation time of CPLEX given in Table 4 is the computation time of CPLEX for solving the fourth subproblem. The computation time of the first three subproblems is even less. What can be concluded is that the computation time of CPLEX is negligible.

In blocks 1 & 2 a requested workload of 47, in blocks 3 & 4 a requested workload of 27 and in blocks 5 & 6 a requested workload of 33 could not be assigned. Especially in blocks 1 & 2 this is caused by the small difference between the requested and offered workload. However, the main causes are preference lists for which it was impossible to assign the requested workload. Some examples of such wrongly chosen preference lists are:

- an empty preference list, because students didn't hand it in on time;
- a preference list with less than ten courses;
- a preference list with not enough different time slots in one of the two blocks;
- a preference list with the same course in multiple positions; there was even a student with the same course ten times on his preference list.

If all students were to hand in a preference list with ten courses and enough different time slots, then in blocks 1 & 2 only five students would not be assigned to their requested number of courses, and in blocks 3 & 4 and blocks 5 & 6 only three students.

Table 4 also shows that in blocks 1 & 2 only 439 out of 590 urgency requests could be assigned. This can be explained by the fact that in these blocks all courses on the preference list of first year students are set as urgent. Most of those preference lists contain six suitable urgent courses of which at most four are assigned. This means at least two not assigned courses with an urgency for each first year student.

The average position denotes the average of the positions of all courses assigned to a student. On average students request a workload of 4, which mostly corresponds with four courses. For example, if courses on the positions 1, 3, 5 and 7 are assigned, then the average position for this student is 4. Hence, it can be concluded that students get a lot of courses which are on top of their preference list.

During the academic year, the average position increases. This can be explained by the fact that in blocks 5 & 6 the first year students are allowed to choose four courses from the same course pool as the other students. In blocks 3 & 4 they were mostly assigned to two out of the six obligatory courses and to two courses chosen by themselves. In blocks 1 & 2 they were preassigned to obligatory courses.

If a student is assigned to  $i \geq 1$  courses from the 7th to 10th position on his preference list he has  $i - 1$  bad positions. From the number of bad positions it can be concluded that students are assigned to courses at the top of their preference lists. That the number of bad positions increases during the year is explained by the first year students.

## 6 Conclusions

We have formulated, analyzed and solved a real-world timetabling problem that showed up at the department of Industrial Design of the TU Eindhoven. Our successful approach was based on an Integer Linear Programming formulation. The running time that CPLEX needs for solving the resulting instances is negligible. An advantage of an Integer Linear Programming approach is its flexibility. Our experience is that the constraints of the timetabling problem change every academic year and even during the academic year.

The administration and the students of the department of Industrial Design were highly satisfied with the timetables generated by our program. Most students now receive courses that are on top of their preference lists. There still are a few students who are not satisfied, but in most cases this turned out to be solely their own fault: they failed to specify correct preferences in the correct format.

## References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, NJ (1993)
2. Avella, P., Vasilev, I.: A computational study of a cutting plane algorithm for university course timetabling. *Journal of Scheduling* 8, 497–514 (2005)
3. Burke, E.K., Kingston, J., Jackson, K., Weare, R.: Automated university timetabling: the state of the art. *The Computer Journal* 40, 565–571 (1997)
4. Burke, E.K., McCollum, B., McMullan, J.P., Qu, R.: Examination timetabling: A new formulation. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 373–375 (August 2006)
5. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operations Research* 140, 266–280 (2002)
6. Busam, V.A.: An algorithm for class scheduling with section preference. *Communications of the ACM* 10, 567–569 (1967)
7. Carter, M.W., Laporte, G.: Recent developments in practical examination timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996)

8. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
9. Cheng, E., Kruk, S., Lipman, M.: Flow formulations for the student scheduling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 299–309. Springer, Heidelberg (2003)
10. Daskalaki, S., Birbas, T.: Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operations Research* 160, 106–120 (2005)
11. de Werra, D.: An introduction to timetabling. *European Journal of Operations Research* 19, 151–162 (1985)
12. Dimopoulou, M., Miliotis, P.: Implementation of a university course and examination timetabling system. *European Journal of Operations Research* 130, 202–213 (2001)
13. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing* 5, 691–703 (1976)
14. Feldman, R., Golumbic, M.C.: Constraint satisfiability algorithms for interactive student scheduling. In: IJCAI 1989. Proceedings of the 11th International Joint Conference on Artificial Intelligence, pp. 1010–1016 (1989)
15. Garey, M.R., Johnson, D.S.: *Computers and Intractability – a Guide to NP-Completeness*. Freeman, San Francisco (1979)
16. Laporte, G., Desrochers, S.: The problem of assigning students to course sections in a large engineering school. *Computational Operations Research* 13, 387–394 (1986)
17. McCollum, B.: A perspective on bridging the gap in university timetabling. In: Burke, E.K., Rudová, H. (eds.) PATAT 2006. LNCS, vol. 3867, pp. 3–23. Springer, Heidelberg (2007)
18. Miyaji, I., Ohno, K., Mine, H.: Solution method for partitioning students into groups. *European Journal of Operations Research* 33, 82–90 (1981)
19. Petrovic, S., Burke, E.K.: University timetabling. In: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, ch. 45, CRC Press, Boca Raton, FL (2004)
20. Sabin, G.C.W., Winter, G.K.: The impact of automated timetabling on universities – a case study. *Journal of Operations Research Society* 37, 689–693 (1986)
21. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
22. Schmidt, G., Ströhlein, T.: Timetable construction – an annotated bibliography. *The Computer Journal* 23, 307–316 (1980)
23. Tripathy, A.: Computerised decision aid for timetabling – a case analysis. *Discrete Applied Mathematics* 35, 313–323 (1992)
24. Zampieri, A., Schaerf, A.: Modelling and solving the Italian examination timetabling problem using tabu search. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 487–491 (August 2006)

## A Max-Flow Model of Problem Formulation No. 1

Problem formulation No. 1 can be modeled as a network flow problem. An illustration of the network with its arc capacities can be found in Figure 1. The layered network has a source node connected to the first layer of nodes of which

each node corresponds to a student. The capacity of the arcs between a student node and the source node is equal to the requested number of courses of the student. For each student and each time slot, a node is defined in the second layer. Each student node has  $|T|$  outgoing arcs with capacity one.

The third layer of the graph contains a node for every course. If a course is on the preference list of a student, this course node is connected with the node of its time slot of the student. For example, in Figure 1 course number 1 is given in time slot 1. Because course 1 is chosen by student 1 and student  $|S|$ , there are arcs from nodes  $t_{11}$  and  $t_{|S|1}$  to course node  $C_1$ . Each course node is connected to the sink with its maximum number of students as its capacity.

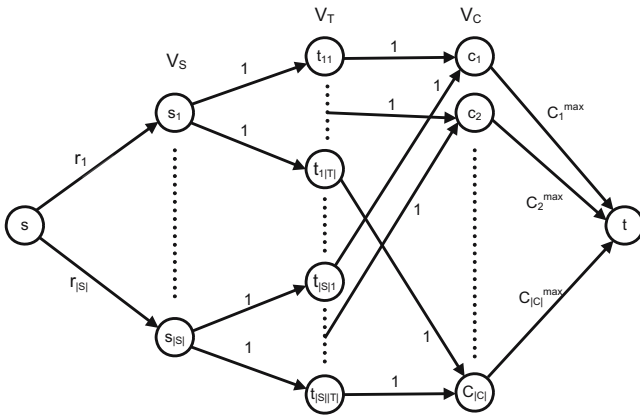


Fig. 1. The network flow model

Note that maximizing the flow through this network maximizes the number of assigned courses and does not deal with the positions of the courses on the preference lists.

## B Some NP-Hardness Results

The timetabling problem defined in Section 3.2 is an NP-hard problem. We prove this by identifying two independent NP-hard subproblems. Both subproblems result from adding one additional constraint to the problem formulation No. 1.

In the first subproblem, the additional constraint are lower bounds on the number of students in the courses. There are no time slots, there is only one section for each course  $c$  with a minimum and a maximum number of participating students. The workload of all courses is one, and only one block has to be scheduled. Formally, problem  $P_{\min}$  is defined as follows:

**Instance:** A set  $C$  of courses; for every course  $c \in C$  a minimum capacity  $C_c^{\min}$  and a maximum capacity  $C_c^{\max}$  of participating students. A set  $S$

of students; for every student  $s \in S$  a preference list of some courses in  $C$ , and a number  $r_s$  of requested courses.

**Question:** Does there exist an assignment such that (i) every student  $s$  gets exactly  $r_s$  courses from its preference list, and such that (ii) for every course  $c$  the number of assigned students is either zero (if the course does not take place) or falls between the bounds  $C_c^{\min}$  and  $C_c^{\max}$ ?

**Theorem 1.** *Problem  $P_{\min}$  is NP-hard.*

*Proof.* The proof is done by reduction from the *exact cover by 3-sets* problem: Given a ground set  $X = \{x_1, \dots, x_n\}$  and a set  $T = \{t_1, \dots, t_m\}$  of 3-element subsets of  $X$ , can one select  $T' \subseteq T$  such that every element of  $X$  occurs in exactly one member of  $T'$ ?

From an instance of the exact cover by 3-sets problem, we construct a corresponding instance of problem  $P_{\min}$  with  $n$  students  $x_1, \dots, x_n$  and with  $m$  courses  $t_1, \dots, t_m$ . Every student  $s$  has a demand of one course ( $r_s = 1$ ), and every course  $c$  has minimum and maximum capacity three ( $C_c^{\min} = C_c^{\max} = 3$ ).

Assume  $X$  possesses an exact cover  $T'$ . Assign student  $x_s$  to course  $t_c$  if and only if  $x_s \in t_c$  and  $t_c \in T'$ . Since  $T'$  is an exact cover of  $X$ , every student  $x_s$  will be assigned to exactly one course  $t_c$ . The course  $t_c$  is assigned to three students if it is in  $T'$ , and to zero students if it is not in  $T'$ . This shows that the constructed instance of  $P_{\min}$  is a yes-instance. The converse statement can be seen in a similar way.  $\square$

In the second subproblem, we take problem formulation No. 1 and additionally allow courses with a workload of 2. We consider a situation with only one section for each course  $c$ , only a single block, and without any time slots. (And there is no minimum capacity of courses.) Problem  $P_{wl}$  is defined as follows:

**Instance:** A set  $C$  of courses; for every course  $c \in C$  a workload  $wl_c \in \{1, 2\}$  and a maximum capacity  $C_c^{\max}$  of participating students. A set  $S$  of students; for every student  $s \in S$  a preference list of some courses in  $C$ , and a desired workload  $r_s$ .

**Question:** Does there exist an assignment such that (i) every student  $s$  gets courses with a total workload  $r_s$  from  $P_s$ , and such that (ii) for every course  $c$  the number of assigned students is at most  $C_c^{\max}$ ?

**Theorem 2.** *Problem  $P_{wl}$  is NP-hard.*

*Proof.* The proof is done by reduction from the 3-SAT variant where every variable occurs exactly twice in negated and exactly twice in unnegated form. Consider an arbitrary instance of this 3-SAT variant:

- For every variable  $x_i$ , we introduce two corresponding students  $st(x_i)$  and  $st(\overline{x_i})$  which both request a workload of two.
- For every variable  $x_i$ , we also introduce a corresponding variable-course  $C(x_i)$  which has a workload of two and a capacity of one.  $C(x_i)$  is in the preference list of  $st(x_i)$  and  $st(\overline{x_i})$ .

- For every clause  $c_j$ , we introduce a clause-course  $C(c_j)$  with a workload of one and a capacity of two. Clause-course  $C(c_j)$  is in the preference list of a student  $st(x_i)$  (respectively  $st(\overline{x_i})$ ) if and only if  $x_i$  (respectively  $\overline{x_i}$ ) occurs as a literal in clause  $c_j$ .

Note that in any feasible assignment, student  $st(x_i)$  (respectively student  $st(\overline{x_i})$ ) will either do course  $C(x_i)$  or the two courses  $C(c_{j1})$  and  $C(c_{j2})$  for which literal  $x_i$  (respectively literal  $\overline{x_i}$ ) occurs in clauses  $c_{j1}$  and  $c_{j2}$ .

Assume that the 3-SAT instance is a yes-instance, and consider a corresponding satisfying truth-assignment. If  $x_i$  is set to TRUE, then we assign student  $st(x_i)$  to the variable-course  $C(x_i)$ , and student  $st(\overline{x_i})$  to the two clause-courses that correspond to the clauses containing  $\overline{x_i}$ . If  $x_i$  is set to FALSE, we assign  $st(x_i)$  to the clause-courses that correspond to the clauses containing  $x_i$ , and student  $st(\overline{x_i})$  to  $C(x_i)$ . Then each student receives his requested workload, and every course  $C(x_i)$  gets only a single student. Since every clause has at most two FALSE literals, the corresponding clause-course will get at most two students. So every yes-instance of the 3-SAT problem leads to a yes-instance of the timetabling problem.

Now assume that the constructed instance of problem  $P_{wl}$  is a yes-instance. Then every student  $st(x_i)$  receives a workload of 2, which implies that the student must either be assigned to one course  $C(x_i)$ , or to two clause-courses  $C(c_{j1})$  and  $C(c_{j2})$ . If student  $st(x_i)$  is assigned to the variable-course  $C(x_i)$ , we set  $x_i$  to TRUE. If student  $x_i$  is assigned to some clause-courses, then we set  $x_i$  to FALSE. Since each clause-course  $C(c_j)$  is assigned to at most two students, every clause contains at most two FALSE literals. Hence, every yes-instance of  $P_{wl}$  corresponds to a yes-instance of 3-SAT.  $\square$

# The Teaching Space Allocation Problem with Splitting

Camille Beyrouthy<sup>1</sup>, Edmund K. Burke<sup>1</sup>, Dario Landa-Silva<sup>1</sup>,  
Barry McCollum<sup>2,3</sup>, Paul McMullan<sup>2</sup>, and Andrew J. Parkes<sup>1</sup>

<sup>1</sup> School of Computer Science and IT,  
University of Nottingham, Nottingham NG8 1BB, UK  
{cbb,ekb,jds,ajp}@cs.nott.ac.uk

<sup>2</sup> Queen's University of Belfast, Belfast, BT7 1NN, UK  
{b.mccollum,p.p.mcmullan}@qub.ac.uk

<sup>3</sup> Realtimesolutions Ltd, 21 Stranmillis Road, Belfast, BT9 5AF  
b.mccollum@realtimesolutions-uk.com

**Abstract.** A standard problem within universities is that of teaching space allocation which can be thought of as the assignment of rooms and times to various teaching activities. The focus is usually on courses that are expected to fit into one room. However, it can also happen that the course will need to be broken up, or ‘split’, into multiple sections. A lecture might be too large to fit into any one room. Another common example is that of seminars or tutorials. Although hundreds of students may be enrolled on a course, it is often subdivided into particular types and sizes of events dependent on the pedagogic requirements of that particular course.

Typically, decisions as to how to split courses need to be made within the context of limited space requirements. Institutions do not have an unlimited number of teaching rooms, and need to effectively use those that they do have. The efficiency of space usage is usually measured by the overall ‘utilisation’ which is basically the fraction of the available seat-hours that are actually used. A multi-objective optimisation problem naturally arises; with a trade-off between satisfying preferences on splitting, a desire to increase utilisation, and also to satisfy other constraints such as those based on event location and timetabling conflicts. In this paper, we explore such trade-offs. The explorations themselves are based on a local search method that attempts to optimise the space utilisation by means of a ‘dynamic splitting’ strategy. The local moves are designed to improve utilisation and satisfy the other constraints, but are also allowed to split, and un-split, courses so as to simultaneously meet the splitting objectives.

## 1 Introduction

An important issue in the management of university teaching space is that of planning for future needs. Support for such decision-making is generally divided into two broad, and sometimes overlapping, areas:



- *space management*: near-term planning,
- *space planning*: long-term planning, including capacity planning.

A fundamental stage of capacity planning aims to estimate the projected student enrollments, and multiply by the expected weekly student contact hours to obtain the total demand for ‘seat-hours’. Similarly, for the rooms we could just sum up the room capacities and multiply by the number of hours they are available in order to determine the ‘seat-hours supply’. A naive way to perform capacity planning, based on such seat-hours estimates, would be simply to ensure that the supply exceeds the demand. However, it is very rare that it is possible to use all of the seats. The efficiency of space usage is usually measured by giving a figure for the ‘utilisation’: i.e., the fraction (or percentage) of available seat-hours that actually end up being used. In real institutions, the utilisation can be surprisingly low, perhaps only 20–50%. To compensate for this, when planning the amount of teaching space to supply, we need to build in excess capacity [13,14].

Naturally, such excess capacity is expensive, because it entails planning for seats to be underused. Good planning should reduce the excess capacity without increasing the risks that expected activities will not find a space. However, this is difficult because there is little fundamental understanding of why the utilisation is so low in the first place, or of the interaction of various constraints and objectives with the utilisation.

A study of this issue was initiated in [5,6]. However, that work, like the majority of work on (university) course timetabling research was concerned with unsplitable ‘events’ (or ‘courses’ or ‘classes’). Such courses are ‘atomic’: i.e. they are not to be subdivided but need to be assigned to a single room and timeslot. However, in some circumstances, courses cannot be taken to be atomic, but must instead be subdivided, or ‘split’, before allocating them to rooms and timeslots. In this paper, we extend the work of [5,6] to the case of courses that require considerable splitting.

Our ongoing investigation into space management and space planning is closely related to research into automated timetabling but we emphasise that there is a crucial difference between the two. In automated timetabling, the set of events that should be accommodated into timeslots and rooms is usually fixed. This means that the space utilisation, in terms of seat-hours demand and offer, is also fixed from the outset. However, in this paper we want to study those factors that have an impact upon space utilisation (even before constructing the timetable). For this, we investigate a scenario in which the seat-hours demand (events to accommodate) is much larger than the seat-hours offer (available rooms). This allows us to vary the utilisation by selecting those events that will be accommodated and those that will be not. We note that although the algorithms presented here allocate events into rooms and timeslots, we are not proposing a timetabling approach. We are presenting a study that helps us to understand the interactions between space utilisation and aspects such as timetabling constraints and others.

Course splitting tends to be driven by one (or both) of the following requirements:

1. *Small-group splitting*: Courses that are intrinsically designed to be taught in small groups, such as seminars or tutorials.
2. *Constraint-driven splitting*: Courses that could, in principle, be held without splitting, but for which splitting is forced because of the following constraints:
  - (a) *capacity constraints*: the course is simply too large to fit into one room.
  - (b) *timetable constraints*: the enrollment is large and across such a wide spectrum of students that it will conflict with many other courses. This greatly reduces the chances of obtaining a conflict-free timetable. Splitting such a course into multiple sections can ease timetabling pressures, as students are more likely to be able to find a section that is conflict-free for them.

Standard university course timetabling methodologies (see [4,7,8,9,10,15,16,18]) assign events to rooms and timeslots, satisfying capacity constraints, so that students do not have to take two events at the same time (and possibly some sequencing or adjacency constraints) and aiming to improve the satisfaction of soft constraints such as the avoidance of unpopular times. The best-known problem that considers ‘timetabling with splitting’ is the ‘student sectioning problem’, e.g. [2,3]. This problem considers the enrollment of students into courses, but each course consists of multiple sections and students need to be assigned to sections in such a way as to avoid timetable clashes whilst respecting room capacities. This means that the student sectioning problem is most relevant to the short period between students enrolling into courses and students needing to know which section they should attend.

However, in this paper, we are not studying such ‘immediate’ space management problems as the student sectioning problem. Instead, we are concerned with decision support for space capacity planning over a longer time frame. For space planning, we need to understand which utilisations are achievable and how they depend on the decision criteria such as section sizes and the constraints arising from location and timetabling. Our goals are:

- Devise algorithms to carry out splitting together with event allocation;
- Explore and understand the trade-offs between the various objectives;
- Understand the impact of such trade-offs on the use of expected utilisation as a safety margin within space planning.

It should be stressed that the splitting algorithms proposed here are used to investigate long-term space planning and not to address near-term space management which is associated to timetabling.

To achieve the above goals, our general approach can be outlined as follows:

1. *Formulate or model the problem*: This includes obtaining a model of splitting that contains the main aspects – although it does not need to contain all the details. For example, we will cover the small group requirements by simply introducing objectives related to the section size or number.

2. Use local search and simulated annealing to explore the solution space and deal with the splitting problem.
3. Carry out experiments in order to visualise the trade-off surfaces.

The specific contributions made in this paper are:

- *Dynamic splitting*: A local search based on exchanges of events, but in which we also make decisions on how to do the splitting. Moves can split courses, and can also rejoin them in order to suit the available rooms.
- *Preliminary trade-off surfaces*: We present results on the interaction of objectives such as location and timetabling, with preferences on section sizes.

*Outline of the paper.* Section 2 gives a basic description of the problem constraints and objective functions and a brief description of the data sets. In Section 3, we outline a form of local search that does not include splitting, but which forms a good basis for the algorithms for splitting presented in Section 4. In Section 5, we compare the performances of the various algorithms. In Section 6, we move to the exploration of the solution space itself, presenting results for the trade-offs between the various objectives.

## 2 Problem Description

Teaching space allocation is concerned with allocating events (courses/course offerings, tutorials, seminars) to rooms and times. In this section, we will cover the basic language of the problem; the constraints and objectives, and the dataset that we will use.

### 2.1 Courses, Events and Rooms

For each *course* we have

1. Size: the number of students in the course.
2. Timeslots: the number of timeslots the course uses during the week.
3. Spacetype: Lecture, Seminar, Tutorial, etc.
4. Department: the department that owns or administers the course.

One can consider other aspects. For example, special features that are imposed by some constraints. However, we shall not consider these here. Also note that the word ‘course’ can mean many different things; ranging from the entire set of classes constituting a degree down to a single class. However, in this paper, we use ‘course’ in the sense of a set of activities of a single type such as a lecture or tutorial, and associated with a single subject. In the case of lectures, the course would be taught by a single faculty member. In general, a ‘course’ might have multiple associated types. For example, lectures in French grammar might always be accompanied by seminars on French literature. However, for the purposes of this paper, we will disregard such cross-spacetype dependencies and regard the lectures and tutorials as separate courses.

Courses will generally be split into sections, though we generally use the term *event* to denote courses/sections that are ‘atomic’: that is, to be assigned to a single room and timeslot. Events have the same information as courses except that each takes only a single timeslot. For events we have

1. Size: Number of students
2. Spacetype: Lecture, Seminar, Tutorial, etc.
3. Department: Department offering/managing the event.

For every *room* we have

1. Capacity: Maximum number of students in the room.
2. Timeslots: The number of timeslots per week.
3. Spacetype: Space for Lecture, Seminar, Tutorial, etc.
4. Department: The one that owns/administers the room.

The basic hard constraints (i.e. those that we always enforce) are

1. *Capacity constraint*: Size of an event cannot exceed the room capacity.
2. *No-sharing constraint*: At most one event is allowed per ‘room-slot’, where by room-slot we refer to a (room,timeslot) pair.

In this paper, we also apply the condition that the spacetype of the event must be the same as that of the room. In general, this hard constraint can be softened, and the resulting spacetype mixing is an important issue, but will be left for future work. So, henceforth, in descriptions of the algorithms we will ignore spacetypes.

## 2.2 Penalty and Objective Functions

Merely allocating events to room-slots so as to satisfy the capacity constraints and no-sharing constraints on its own is not useful; we also need to take into account space utilisation objectives for additional soft constraints. Based on the work in [5,6], and also from considerations of what a good allocation is likely to mean in the presence of splitting, we use the following:

**Utilisation (U).** [5,6] The primary objective is that we want to make good use of the rooms, and have a good number of student contact hours. We will measure this by the ‘Seat-Hours’ – which is just the sum over all rooms and timeslots of the number of students allocated to that room-slot. The utilisation  $U$  is then defined as just the Seat-Hours achieved as a fraction of the total Seat-Hours available (the sum over all rooms and times of the room capacity):

$$U = \frac{\text{Seat-Hours used}}{\text{total Seat-Hours available}}. \quad (1)$$

This is usually expressed as a percentage:  $U = 100\%$  if and only if every seat is filled at every available timeslot.

**Timetabling (TT).** [5,6] Teaching space allocation is also constrained by timetabling needs, and we take this aspect into account. Hence, we use here a timetabling penalty (TT) that is just a standard conflict matrix between events which represents pairs of events that should not be placed in the same timeslot. For this paper, we will simply use randomly generated conflict graphs. We use  $TT(p)$  to denote that each potential conflict is taken independently with probability percentage,  $p$ . For example,  $TT(70)$  means that the conflict density is (about) 70%.

*Conflict Inheritance Problem.* Course conflicts are used to represent the case that students are enrolled for both of the courses in the conflict. In standard university timetabling, the conflict graph will be fixed, but with sectioning. Part of the point is that students can be assigned to sections with the intention of resolving conflicts. The problem of assigning students to sections is treated, for example, in [2,3,12]. In [3] a relaxed conflict matrix is created and, in particular, it is less dense than the matrix between courses. Hence, if a course has multiple sections, then not every section ought to have the same conflicts as the parent course. That is, there is a ‘conflict inheritance problem’: when a course is split, how should we decide upon the timetable conflicts given to the resulting events (also see [17])? This problem is not studied here, but it represents a promising direction for future work. In this initial study of splitting, we will look at the simpler case in which the inheritance is full; that is, on splitting, each event inherits *all* the conflicts of the course.

**Location (L).** [5,6] A common objective in timetabling is the goal of reducing the physical travel distances for students between events. It also seems likely that students and faculty would prefer that the events they attend will be close to their own department. We do not attempt to model this exactly but instead use a simple model in which there is a penalty if the department of the event is different from that of the room-slot. Specifically, if an event  $i$  has department  $D(i)$ , and is allocated to a room  $r$  with department  $D(r)$ , then there is a penalty matrix derived from the department,  $Y(D(i), D(r))$ . Events in their own department are not penalised,  $Y(d, d) = 0$ , and the off-diagonal elements were selected arbitrarily (as we did not have physical data). The total Location penalty is just the sum of this penalty over all allocated events.

**Section Size (SZ).** For courses such as tutorials or seminars it is standard that they are intended to be in small groups. Hence, when splitting we need to be able to control the sizes of the sections. In this paper, we use a simple model in which we take a target size for the sections, and simply penalise the deviation from that target. Given an allocated event  $i$ , let the number of students be  $c_i$ , the total number of allocated events be  $I$ , and the target section size  $T$ . The section size penalty SZ that we use is

$$SZ = \sum_{i=1}^I |c_i - T|. \quad (2)$$

**Section number (SN).** Every section will need a teacher, and so the total number of sections allocated will have a cost in terms of teaching hours, and should not be allowed to become out of control. The penalty SN is simply the total number of allocated events. Pressure to minimise SN will tend to discourage courses from splitting into more events than are needed.

**No Partial Allocation (NPA).** The context in which we undertake the search is that we have a large pool of courses available and are investigating the best subset that can be allocated. However, if a course is broken into sections, then the course as a whole ought to be allocated or not. The NPA penalises those cases in which some of the sections of a course are allocated, but other events from the same course remain unallocated. Enforcing NPA as a hard constraint would disallow partial allocation: for every course, either all sections are allocated, or none are allocated.

### 2.3 Overall Objective Function

The overall problem is a multi-objective optimisation problem because there is conflict between improving utilisation and satisfying the constraints. However, we use a linearisation into a single overall objective or fitness  $F$ , which can be represented as follows:

$$F = W(U) \cdot U + W(L) \cdot (-L) + W(TT) \cdot (-TT) \\ + W(SZ) \cdot (-SZ) + W(SN) \cdot (-SN) + W(NPA) \cdot (-NPA) \quad (3)$$

where the  $W(*)$  are simply weights associated with each objective or penalty. The minus signs merely change penalties into objectives and make all the ‘dimensions’ or objectives into maximisation problems.

The aim is to maximise  $F$  and consequently maximise utilisation ( $U$ ) while reducing the penalties for  $L$ ,  $TT$ , etc. In practice, we will consider a wide variety of relative weights. Of course, if a weight is large enough then it effectively turns the penalty into a hard constraint. The use of weights is also intended to allow modelling of the way that administrators will relax some penalties and tighten others.

### 2.4 Datasets

Table [1](#) gives an overview of the four datasets we use to test our splitting algorithms. All datasets are collected from a building of a university in Sydney, Australia. (We omitted the ‘lectures only’ dataset used for [\[5,6\]](#) as it is not relevant to splitting). Note that we use datasets in which the demand of seat-hours is much larger than their supply because this is the case that is relevant to our study. Improvements or detriments on space utilisation can only occur when the subset of events that are allocated changes.

The workshops dataset,  $Wksp$ , is mainly characterized by the non-uniform capacity of rooms ranging from 21 to 80, making it possible for some small

**Table 1.** The four datasets that we use, and some of their properties, including numbers of rooms and courses, the total *Seat-Hours* demanded by all the courses, and the *Seat-Hours* available in all the rooms

Data-set	Wksp	Tut	Sem	Tut-trim
Spacetype	Workshop	Tutorial	Seminar	Tutorial
No. of courses	1077	2088	3711	620
No. of rooms	16	184	88	47
Timeslots no.	48	46	46	50
Seat-Hours, courses	86,140	290,839	440,131	87,678
Seat-Hours, rooms	39,408	163,500	176,318	41,350

courses to fit without splitting. For **Tut**, the main characteristic of this dataset is the small capacity of rooms and their uniformity, e.g. most rooms have sizes in the range 8–20; enforcing a section size is therefore trivial in this case. The full dataset, **Tut**, is quite large and so, in order to be able to plot trade-off surfaces in a reasonable amount of time, we also created the set **Tut-trim** by randomly selecting a fraction of the rooms and courses. The seminar dataset, **Sem**, is similar in structure to **Tut**. It exhibits the same characteristics as **Tut**, and has room capacities ranging from 30 to 86 students. Both seminars and tutorials have relatively large courses and therefore splitting is essential for them.

### 3 Algorithms Without Splitting

In this section, we present the methods we use for cases when splitting is neither needed nor performed. Although, the focus of the paper is on splitting we think that describing the non-splitting local operators first helps the presentation of the paper.

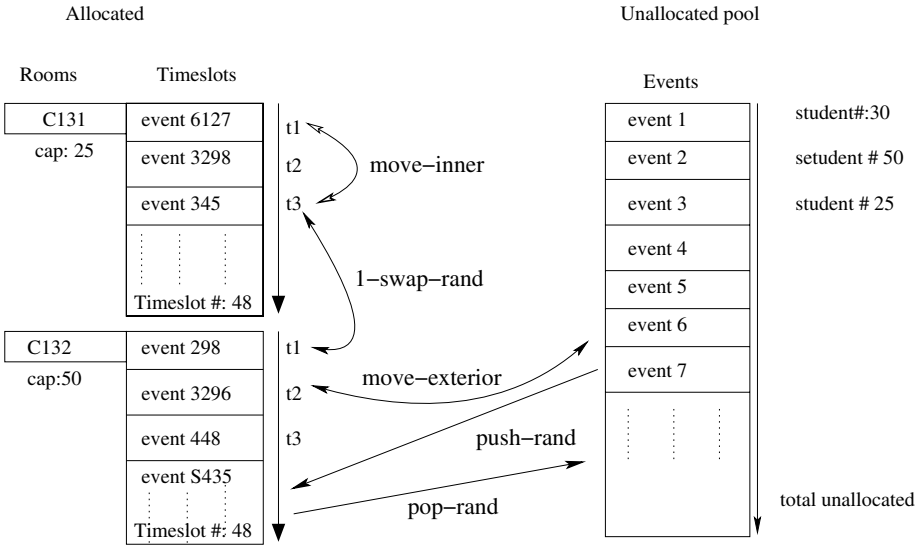
#### 3.1 Local Search Operators Without Splitting

The neighbourhood moves used to explore the search space are given below. Note that, by construction, all operators (implicitly) maintain feasibility of the solution. Figure 1 illustrates these local search operators.

**1-swap-rand:** Randomly select 2 different rooms and, in each room, randomly select an allocated event. The selected events are swapped between rooms. If the given events violate any of the hard constraints, we randomly search again for 2 other events to swap.

**2-swap-rand:** Similar to *1-swap* but it randomly selects 4 (2 from each room) rather than 2 events and swaps them. Special consideration is given to checking that the 4 events are all different and that one swap would not cancel the other.

**Move-exterior:** Randomly selects an allocated and an unallocated event and tries to swap them; assigning the unallocated event to the timeslot of the allocated one.



**Fig. 1.** Schematic of the local search operators (except *2-swap-rand*) for the local search without splitting

**Push-rand:** Randomly selects one course from the unallocated set of events and tries to allocate it to a randomly selected room, also picking the timeslot at random.

**Push-rand-p:** This move is another version of *push-rand* but which gives priority to early timeslots in the rooms timetable, favouring them over late ones. The local search is allowed to switch probabilistically between the 2 different versions of *push-rand*.

**Pop-rand:** Randomly selects one event from a randomly selected room and deallocates it.

**Move-inner:** Swap 2 randomly selected events in a given room between 2 randomly selected timeslots.

### 3.2 Meta-heuristics

We only use hill-climbing and simulated annealing [11] implementations in this paper.

The hill climbing algorithm (HC) variant uses most of the moves given above to perform a search of the neighbourhoods. On each iteration, it selects an operator from the list above according to a given move probability and applies it to generate a candidate solution. If the candidate solution has better (or equal) fitness than the incumbent, we commit to the move, but otherwise disregard it.



Simulated Annealing (SA) was used as the main component for overcoming local optima. A geometric cooling schedule was used, specifically temperature  $T \rightarrow \alpha T$  every 650 iterations with  $\alpha = 0.998$ . We generally used 6 million iterations and initial temperature  $T_i = 0.6$ . Such a slow cooling and such a large number of iterations were chosen to err on the side of safety.

## 4 Algorithms with Splitting

In this section, we describe the splitting heuristics that are incorporated into the HC and the SA approaches. Two strategies are implemented: (a) construction-based splitting, and (b) dynamic local search-based splitting. In the first case, the section size is calculated during the construction of an initial solution and remains fixed for all events throughout the local search. In dynamic splitting, the section size is calculated as the local search progresses according to the size of the event (and room capacity) that is being allocated. Hence, we will have

- SS-HC: Construction-based static splitting and hill-climbing
- SS-SA: Construction-based static splitting and simulated annealing
- DS-HC: Dynamic splitting and hill-climbing
- DS-SA: Dynamic splitting and simulated annealing.

### 4.1 Static Splitting

In static splitting we select a target section size (generally based on room profiles) and then split all the courses of size larger than that target size, into as many sections as needed during the process of constructing an initial solution. We use the term static, because once a split is enforced it cannot be changed. We afterwards run a local search algorithm (HC or SA) to improve the initial solution. So, in this strategy, splitting happens within the construction and this provides no flexibility in changing section size during the local search.

There can be many ways to calculate and fix the target section size. Here we compare three variants which are based on the notion of a ‘target room capacity’. This means that the target section size is calculated based on the capacity of the rooms that are available for allocating course sections. Specifically, the target section size is fixed to one of three different values:

1. MAXCAP – the largest room capacity
2. AVGCAP – the average room capacity
3. MINCAP – the smallest room capacity.

We recognise that more elaborate ways to calculate the target section size are possible based on information from the room profiles. However, our interest here is to explore how splitting during the construction phase affects the search process in general, and to compare it to the case in which splitting is carried out during the local search (dynamic) which is described in the next section.

## 4.2 Dynamic Splitting Operators

In dynamic splitting, we calculate the section sizes during the local search itself. The dynamic splitting heuristic is also capable of un-splitting/rejoining sections and this gives more flexibility to determine an adequate target section size by changing, adding, deleting and merging sections as needed.

Dynamic splitting is embedded in the local search in such a way that there is freedom and diversity in the choices of section sizes. Thus, the dynamic splitting operators consider not only the room capacities (as in the case of static splitting) but also the location (L), timetabling (TT), section number (SN), section size (SZ), and no partial allocation (NPA) constraints. Note that, at the current stage, the operators themselves do not directly respond to penalties that are considered by the local search. Presumably, this leads to inefficiencies because good moves will need to be discovered via multiple iterations of the SA/HC rather than directly and heuristically with the operators; we intend to investigate this in future work.

In the search, it is important to note that the ‘pool of unallocated courses’ is a pool of the portions of courses that are not yet allocated. The unallocated portions contain no information about how they are going to be split. That is, it is not a pool of sections waiting to be allocated, but instead the sections are created during the process of allocation. That is, the main characteristic of the splitting operators lies in the fact that when a split occurs, we actually select a fraction of a course and allocate it. When a section is unallocated, we merge it back with the associated course without keeping track of previous section splits.

Below, we detail the neighbourhood operators used in the dynamic splitting (ordered roughly by their degree of elaboration):

**1-swap-rand-sec:** This operator works in a similar way to *1-swap-rand* described in section 3.1 but the move is carried out between 2 sections (not necessarily of the same course).

**Move-inner-sec:** This operator works in a similar way to *move-inner* described in section 3.1 but the move is carried out between 2 sections (not necessarily of the same course).

**Push-rand:** This operator works in a similar way to *push-rand* described in Section 3.1 but note that the events being ‘pushed’ to the allocation are sections of a course that are smaller than the chosen room, and so no splitting was needed.

**Pop-unsplit:** This operator is used to remove sections from their allocated room and unsplit/rejoin sections with their unallocated parent course. Note that this move can be seen as the reverse operation to splitting but not exactly, because we do not keep track of the splits made during the search by *split-push* and *split-max* that we describe next. First, the

*pop-unsplit* operator chooses (at random) an allocated event from a randomly selected room. In the case that the chosen event is a section, the operator unallocates the section and merges it with its unallocated parent event. If the event is not a section it is simply added to the unallocated pool.

**Split-push:** This operator is used to handle courses whose unallocated portion is larger than the chosen room. This is the main operator that is used to create new sections. It is at the heart of the dynamic splitting:

**Proc: split-push**

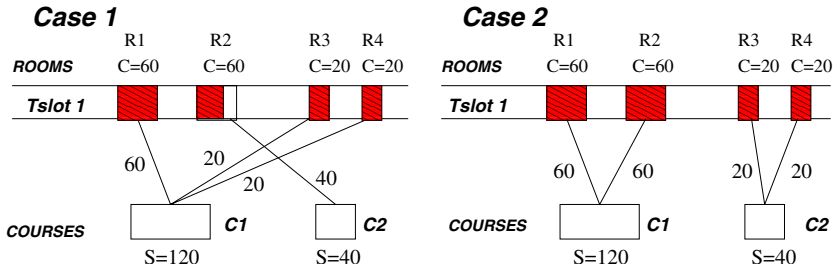
- 1 Randomly select a room  $R_j$  with available timeslots.  
Let its capacity be  $C_j$ .
- 2 Randomly select a course  $P_i$  from the unallocated pool.  
Let the size of  $P_i$  be  $N_i$ .
- 3 Set size  $s = \text{floor}(C_j * \text{rand}(\delta, 1))$   
though if  $s > N_i$  then  $s=N_i$
- 4 Randomly select empty room-slot  $t_j$
- 5 Create section  $S_i$  with size  $s$   
and resize the remainder  $P_i$
- 6 Set that  $S_i$  inherits all conflicts from course  $P_i$  (see section 2.2)
- 7 Generate candidate move by allocating  $S_i$  to room  $R_j$  in timeslot  $t_j$

Note that  $\text{rand}(\delta, 1)$  means a number randomly selected from the interval  $[\delta, 1]$  and the parameter  $\delta$  is described below. After randomly selecting a room-slot and unallocated course, the main step in this operator comes in its decision as to how to split the course to create a new section. Assuming that the capacity of the room is smaller than the size of the remainder of the course, the new section size,  $s$ , is calculated by multiplying the capacity of the room by a randomly selected factor. The factor depends on a ‘section re-sizing parameter’,  $\delta$ , that we give a value between 0.4 and 0.6. Suppose that we take  $\delta = 0.4$  then this effectively means that the generated section size,  $s$ , will be between 40% and 100% of the selected room’s capacity. The intention of this randomised selection of section size is that it enables the search to discover section sizes that match the penalties such as section size and section number. The new section inherits all of the conflict information from its parent course – see the discussion of the ‘Conflict Inheritance Problem’ in Section 2.2. The new section is then allocated to the chosen room. The remaining part of the parent course is left in the unallocated list of courses with its size reduced appropriately.

**Split-max:** This operator is a version of **split-push** with  $\delta=1$  and is designed so that courses with size larger than the chosen room are split so that sections are of the maximum size allowed within the chosen room.

### 4.3 Example of the Operator Application

An example of the search process, and the differences that can arise during search, are illustrated in the simple example of Figure 2. Two courses C1 and C2, of sizes 120 and 40 respectively, are to be allocated to the four rooms available. We have selected capacities so that total size of courses precisely equals the total capacity of the rooms. In the first case, it happens that the smaller event C2 is allocated first via a **push-rand** because it can be allocated to that room without a split. However, this inevitably means that 20 spaces within room R2



**Fig. 2.** Example in which applying operators to split courses has different effects. In case 1, course C2 first receives a *push-rand* into room R2, and then applications of *split-push* to C1 are able to allocate only  $60 + 20 + 20 = 100$  students rather than the needed 120. However, in case 2 we see that reversing the order allows all of both courses to be allocated.

are wasted, and so it becomes impossible to allocate all of course C1. In the second case, the larger course, C1, is first split using *split-max* and then we end up with a perfect fit. The operator *split-max* with its implicit ‘maximum size sections first’ is often better at maximising the utilisation, although there are other cases in which *push-rand* is necessary. For this reason, and also from experimental evidence, we tend to give the operator *split-max* more probability of being selected than the operator *push-rand*.

#### 4.4 Controlling the Search

We have observed, in an informal manner, that the effectiveness of each operator varies during the search. As an example, suppose we are just carrying out non-splitting local search from Section 3. We start with an empty allocation, and then the *Push-rand* operator is the most important and successful in the early stages as events/courses need to be allocated, but for capacity reasons it remains stalled during the rest of the search, during which the other moves provide the bulk of the successful search efforts. This led to us taking a simple, though adequate, compromise with probabilities of around 10–20% for each operator.

## 5 Experimental Comparison of the Algorithms

In this section, we first investigate the ‘static splitting’ method in which only the construction does any splitting and is followed by simulated annealing. Note that ‘construction followed by hill climbing’ is not presented as, unsurprisingly, it performs no better than the simulated annealing version. We find that it is far inferior to the dynamic splitting. Moving to the dynamic splitting itself we then compare the HC and SA variants, and we see that the DS-SA variant is the better.

Although it seems evident from the datasets in Table 1, we illustrate the importance of splitting in our scenario. The following table compares some

examples of the utilisation percentages obtained (and the number of events allocated) without any splitting (not even static splitting from the construction) and compares them with those obtained by DS-SA:

	Wksp	Tut	Sem
SA, no splitting	36% (264 ev)	0.015%	0.013%
DS-SA	70% (720 ev)	26% (1747 ev)	44% (3000 ev)

We clearly see that splitting is essential for the tutorials and seminars as, otherwise, virtually nothing is allocated. For the workshops, some courses can be allocated, but we still lose a lot compared to when splitting is allowed. So from now on we always permit splitting (we refer the reader back to Section 2.4 where the datasets are presented and the difference between the **Wksp** data set and the others was also noted). While, in the results above, utilisation figures seem a little higher than in real-world cases (30–40%) we show, in later sections, how the different actual constraints drive the utilisation down to more practical levels; the introduction of section size penalty along with the No-Partial-Allocation penalty can also generate a realistic level of utilisation figures.

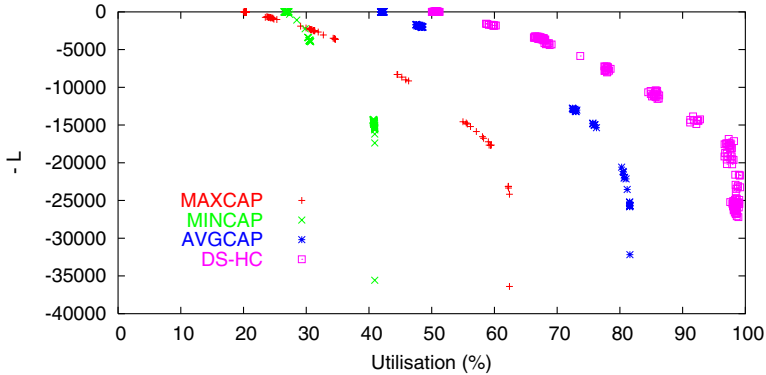
Our results are generically presenting trade-off curves which are approximations to Pareto fronts. These are generally representing the trade-off between two of the objective functions. We select a wide range of relative values for the weights associated with the two chosen objectives, and then call the local search with those weights. For example, we often plot the trade-off between utilisation,  $U$ , and location,  $L$ . In this case, we pick a non-zero value for  $W(U)$ , and then just search at each of many values for  $W(L)$ . This leaves some gaps in the curves due to the presence of unsupported solutions. However, generally the gaps are small and we do not expect that filling them would significantly change the overall messages from the results. Note that since  $L$  is a penalty, then the objective is essentially  $-L$ , and we use this for the  $y$ -axis, so that ‘better’ is towards the top-right corner (and similarly for all others of our trade-off graphs).

## 5.1 Dynamic vs. Static Splitting

Figure 3 shows the trade-off curves between utilisation and location for the three different methods from the static splitting (see Section 4.1), and compares them to the results from the dynamic splitting method, DS-HC.

We see that for the construction, splitting based on the average room capacity (AVGCAP) outperforms the other two (MINCAP and MAXCAP). This is reasonable, as when splitting by the smallest room capacity there is capacity wastage in larger rooms and when splitting is based on the larger room size there is a wastage caused by violating room capacities, since we cannot allocate a section to a room with smaller capacity.

However, it is also clear that all our construction-based splitting methods are easily outperformed by the dynamic splitting. This is unsurprising, as it is



**Fig. 3.** Comparison of dynamic and static (construction-based) splitting for the *Wksp* data set. Plots give the trade-offs obtained between Utilisation and Location, all the other objectives being disregarded ( $W_{TT}=W_{SZ}=W_{SN}=W_{NPA}=0$ ). The first three sets of points are from the three constructive methods of Section 4.1; the last ‘DS-HC’ from the dynamic splitting with hill-climbing.

entirely reasonable that it is best to do splits based upon the availability of room capacities rather than on a uniform target capacity. It is possible that a more sophisticated constructive method would perform much better. However, for the purposes of this paper we will henceforth consider only dynamic splitting.

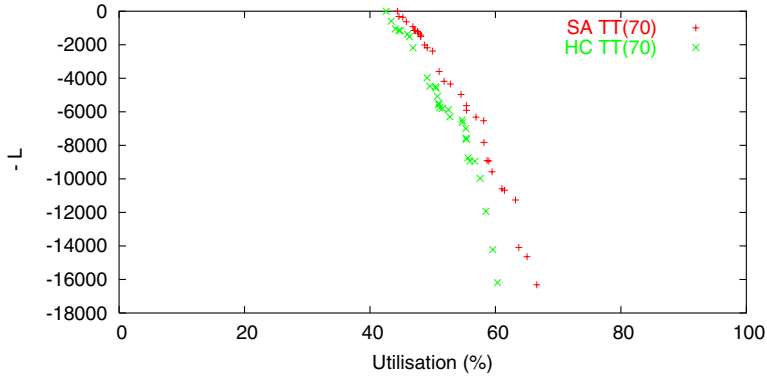
### 5.2 Dynamic Splitting: HC vs. SA

Figure 4 illustrates the different performances of DS-HC and DS-SA on the workshop problems in the presence of timetabling. Figure 5 is the same except that it is for a tutorials dataset. As is well known, the conflict graph of the timetabling penalty moves the problem to a variant of graph colouring. So it is not surprising that the SA is likely to outperform the HC, as SA can escape local minima but HC cannot. Perhaps more surprising is the observation that the performances in the absence of TT are often very similar.

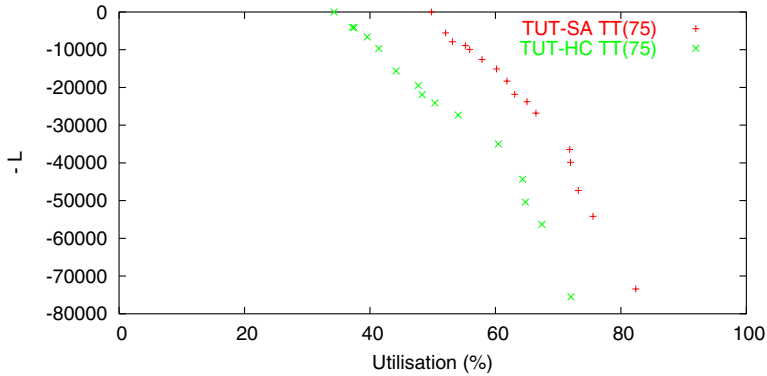
In any case, it is clear that DS-SA is the best of the algorithms that we have considered, and so will be assumed from now on whenever we have a TT penalty (and in the absence of a TT penalty it seemed to matter little which one is used).

## 6 Trade-Offs Between the Various Objectives

Having selected dynamic splitting as our algorithm of choice, we now change focus: we no longer pursue the solution algorithm itself, but instead focus on the solution space. In particular, we present some preliminary results on how the various objectives interact and, in particular, the magnitude of their effect on the utilisation.



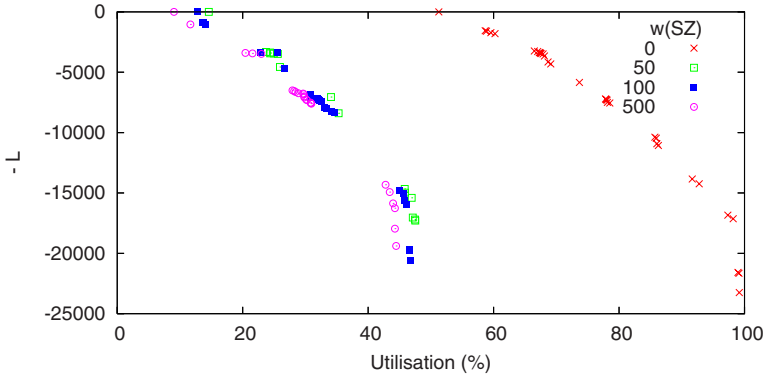
**Fig. 4.** Trade-off of utilisation and location as obtained with dynamic splitting, and using the HC and SA algorithms. For the *Wksp* data, and in the presence of TT(70), and no other constraints beside U, L, and TT.



**Fig. 5.** As Figure 4 but instead using the tutorials dataset, *Tut-trim*, and with TT(75)

### 6.1 Interaction of Section Size Penalty (SZ), Location Penalty (L), and Utilisation (U)

Figure 6 gives plots of the trade-off between utilisation (U) and location (L), in the presence of various weights,  $W(SZ)$ , for the section size penalty (SZ), with a target section size of 25, but with no other penalties. Note that the case  $W(SZ) = 0$  was seen previously as the best line in Figure 3, and illustrates that (even without section size constraints) demanding a low location penalty has the potential to significantly reduce the utilisation (from about 98% down to 50%). The non-zero values for  $W(SZ)$  drastically reduce the utilisation: dropping to the range 10–50%. This corresponds to a policy of a fixed size, but with such an excessively-strict adherence to that policy that the overall room usage suffers.



**Fig. 6.** Trade-off surfaces for the given values of the weight  $W(SZ)$  for the section size policy. On the **Wksp** dataset, with a target section size of 25, and aiming to optimise only utilisation  $U$ , location  $L$ , and section size  $SZ$ .

### 6.2 Trade-Offs Arising from Section Size Penalty and Utilisation

So far, we have only looked at trade-offs between Utilisation and Location. However, now, in Figure 7 we show the trade-off between utilisation,  $U$ , and section size penalty ( $SZ$ ). This happens to be with a small weight given to the section number penalty,  $SN$ ; however, with no other penalties:  $W(L) = W(TT) = W(NPA) = 0$ , so in this case location penalties are ignored. Each curve illustrates the drastic drop in utilisation as we move towards the section size becoming a hard constraint. We also see that reducing the target for the section size reduces utilisations though by a lesser amount. Part of this effect is possibly because our current section size penalty does not allow a range of values for the section size and because it penalises under-filling a section just as much as overfilling.

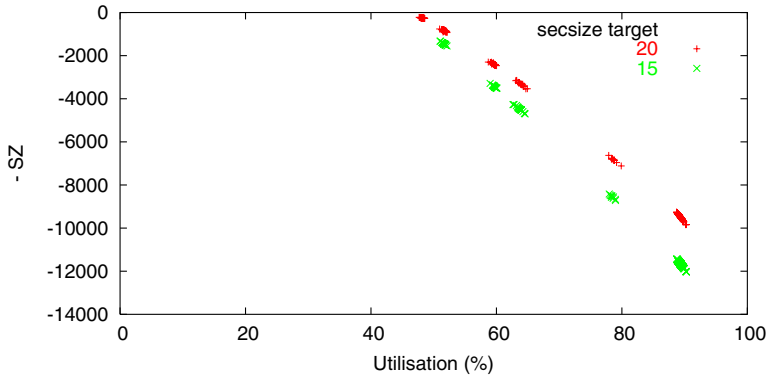
### 6.3 Effects of Timetabling Constraints

Figure 8 is a plot of the usual trade-off between utilisation and location objectives, but comparing the presence and absence of a timetabling constraint. The case with timetabling is with conflict matrix of density 70%, and with an associated weight  $W(TT)$  that is large enough that the timetabling is effectively enforced as a hard constraint. This illustrates that timetabling issues have the potential to significantly reduce the utilisation, and so again could be part of the explanation for the low values of utilisation observed in real problems.

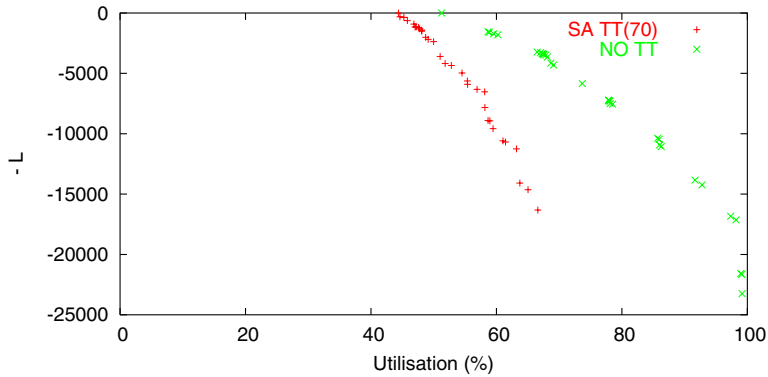
### 6.4 Inclusion of the No-Partial-Allocation Penalty

So far, we have presented results for cases in which the ‘No Partial Allocation’ ( $NPA$ ) objective is ignored: that is,  $W(NPA) = 0$ . This means that some sections from a course can be allocated even though others are unallocated. This gives the search extra freedom, and so it is reasonable that enforcing  $NPA$  will only further





**Fig. 7.** Utilisation vs. section size penalty, SZ, for the Wksp data set, and for two values (15 and 20) of the target section size

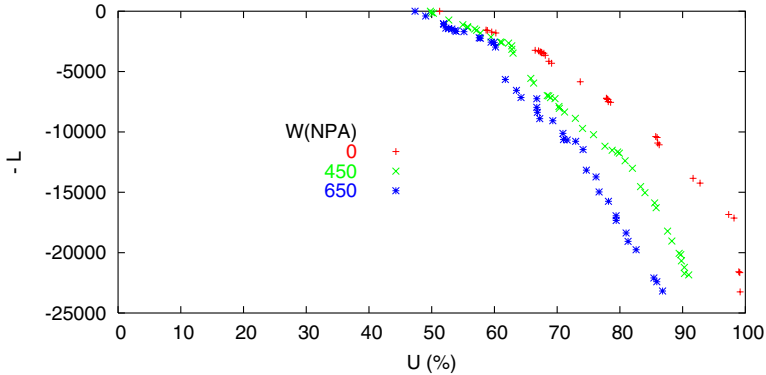


**Fig. 8.** Trade-offs between Utilisation and Location for the Wksp dataset. ‘No TT’ means that no objectives besides L and U are weighted, in particular  $W(TT) = 0$ . In contrast, ‘TT(70)’ means that a timetabling constraint with a density of 70% is enforced as a hard constraint.

reduce the utilisations obtained. The magnitude of this effect is seen in Figure 9. We see that giving NPA high weights can further reduce the utilisation by about 10–20%. This is a significant effect, although it is somewhat smaller than the effects seen in the trade-offs with the timetabling and section size objectives. It is also interesting that the effect of the NPA becomes very small when selecting solutions with small location penalty.

## 7 Summary and Future Work

We have devised methods, and performed preliminary studies, to support long-term teaching space planning in the presence of courses that will need to be split down into multiple sections.



**Fig. 9.** Trade-offs between Utilisation and Location, in the presence of various strengths of the ‘No Partial Allocation’ (NPA) penalty, but with no TT or other penalties

The work has two broad aspects. Firstly, we provided algorithms to perform splitting and optimisation in the presence of multiple objective functions, including overall space usage, constraints inspired from timetabling, and also objectives relating to desirable properties of the splits themselves. In particular, we devised a splitting algorithm in which the decisions as to course splitting are incorporated within a local search.

Secondly, we used an implementation of the dynamic splitting in order to explore the trade-offs between various objectives. We found that the incorporation of objectives other than solely employing utilisation can result in the utilisation dropping from over 90% down to much lower figures such as 30–50%. This is significant because such low utilisations are consistent with the real world; and so our model ultimately has the potential to explain real-world utilisation figures. The intended longer term consequences of such better understanding will enable an improved ability to engineer the safety margins that need to be built into capacity planning. We acknowledge that other factors apart from those considered in this study might also have an impact on the utilisation inefficiency that occurs in real-world problems. The evidence presented here is a first step towards a wider investigation of this issue.

In future work, we intend to improve the speed and scope of the methods. This will have multiple aspects, of which perhaps the most important is to model the conflict inheritance issues that we discussed in Section 2.2. At the moment, we do not answer, or indeed model this problem. In the absence of a good model for this inheritance, we do not answer here the questions as to how the degree of inheritance affects results. Our inheritance is either total or nonexistent. That is, all sections inherit either all conflicts of the associated course, or else they inherit none (equivalent to simply turning off the timetabling penalty). Although a deficiency, this does at least allow us to put bounds on the effect of the timetabling. The effect of partial inheritance must lie between the two extremes of total and no inheritance. Building a model for the partial inheritance, and exploring its effects is a high priority for future work.

## References

1. Aarts, E., Korst, J.: *Simulated Annealing and Boltzman Machines*. Wiley, New York (1990)
2. Alvarez-Valdes, R., Crespo, E., Tamarit, J.: Assigning students to course sections using tabu search. *Annals of Operations Research* 96, 1–16 (2000)
3. AminToosi, M., Yazdi, H.S., Haddadnia, J.: Feature selection in a fuzzy student sectioning algorithm. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 147–160. Springer, Heidelberg (2005)
4. Bardadym, V.: Computer-aided school and university timetabling: The new wave. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 22–45. Springer, Heidelberg (1996)
5. Beyrouthy, C., Burke, E.K., McCollum, B., McMullan, P., Landa-Silva, D., Parkes, A.: Towards improving the utilisation of university teaching space. Technical Report NOTTCS-TR-2006-5, School of Computer Science & IT, University of Nottingham (2006)
6. Beyrouthy, C., Burke, E.K., McCollum, B., McMullan, P., Landa-Silva, D., Parkes, A.: Understanding the role of UFOs within space exploitation. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 359–362 (August 2006)
7. Burke, E.K., Jackson, K., Kingston, J., Weare, R.: Automated timetabling: The state of the art. *The Computer Journal* 40, 565–571 (1997)
8. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002)
9. Carter, M.: Timetabling. In: Gass, S., Harris, C. (eds.) *Encyclopedia of Operations Research and Management Science*, pp. 833–836. Kluwer, Dordrecht (2001)
10. Carter, M., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* 220, 671–680 (1983)
12. Laporte, G., Desroches, S.: The problem of assigning students to course sections in a large engineering school. *Computers and Operations Research* 13, 387–394 (1986)
13. McCollum, B., McMullan, P.: The cornerstone of effective management and planning of space. Technical Report, Realtime Solutions Ltd (2004)
14. McCollum, B., Roche, T.: Scenarios for allocation of space. Technical Report, Realtime Solutions Ltd (2004)
15. Petrovic, S., Burke, E.K.: University timetabling. In: Leung, J. (ed.) *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pp. 1–23. Chapman and Hall/CRC Press, London (2004)
16. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 18–27 (1999)
17. Selim, S.: Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem. *The Computer Journal* 31, 76–82 (1988)
18. de Werra, D.: An introduction to timetabling. *European Journal of Operational Research* 19, 151–162 (1985)

# Solving the University Timetabling Problem with Optimized Enrollment of Students by a Self-adaptive Genetic Algorithm

Radomír Perzina

Department of Mathematical Methods in Economics,  
School of Business Administration, Silesian University,  
University Sq. 1934/3, 733 40 Karviná, Czech Republic  
perzina@opf.slu.cz

**Abstract.** The timetabling problem is well known to be an NP-complete combinatorial problem. The problem becomes even more complex when addressed to individual timetables of students. The core of dealing with the problem in this application is a timetable builder based on mixed direct–indirect encoding evolved by a genetic algorithm with a self-adaptation paradigm, where the parameters of the genetic algorithm are optimized during the same evolution cycle as the problem itself. The aim of this paper is to present an encoding for self-adaptation of genetic algorithms that is suitable for timetabling problems. Compared to previous approaches we designed the encoding for self-adaptation of not only one parameter or several ones but for all possible parameters of genetic algorithms at the same time. The proposed self-adaptive genetic algorithm is then applied for solving the real university timetabling problem and compared with a standard genetic algorithm. The main advantage of this approach is that it makes it possible to solve a wide range of timetabling and scheduling problems without setting parameters for each kind of problem in advance. Unlike common timetabling problems, the algorithm was applied to the problem in which each student has an individual timetable, so we also present and discuss the algorithm for optimized enrollment of students that minimizes the number of clashing constraints for students.

## 1 Introduction

Genetic algorithms are search algorithms based on the idea of natural selection and natural genetics. It is well known that the efficiency of genetic algorithms strongly depends on their parameters. These parameters are usually set up according to vaguely formulated recommendations of experts or by the so-called two-level genetic algorithm, where at the first level the genetic algorithm optimizes parameters of the second level. Self-adaptation seems to be a promising feature of genetic algorithms, whereby the parameters of the genetic algorithm are optimized during the same evolution cycle as the problem itself. The aim of this paper is to present an encoding and genetic operators for self-adaptation of

genetic algorithms that is suitable for solving the university timetabling problem. Compared to previous approaches (e.g. [5, 17, 25]) we designed the encoding for self-adaptation for not only one parameter but for all or nearly all possible parameters of genetic algorithms at the same time. Moreover, the parameters are encoded separately for each element of a chromosome.

The proposed self-adaptive genetic algorithm is then applied for solving the real university timetabling problem at Silesian University. The problem is known to be NP-complete and hence there is no known algorithm for solving it in polynomial time [12]. The requirements for timetabling differs from university to university, but in general the timetabling problem consists of assigning each lecture from a set of lectures to a suitable room and a time slot subject to a number of hard and soft constraints, such as no teacher can teach more lectures at the same time, at no room can be taught more than one lecture at the same time, teachers time and room preferences, etc.

At some universities, including Silesian University, each student has an individual timetable, i.e. there are no groups of students which have the same timetable, and it is even difficult to find two students with the same timetable; thus solving the problem becomes very complex. In order to be able to deal with individual timetables of students we designed an algorithm for optimization of enrollment of students that effectively decreases the number of constraints for student clashes.

A large number of diverse methods have been already proposed in the literature for solving timetabling problems. These methods come from a number of scientific disciplines like Operations Research, Artificial Intelligence, and Computational Intelligence and can be divided into four categories:

1. Sequential methods that treat timetabling problems as graph problems. Generally, they order the events using domain-specific heuristics and then assign the events sequentially into valid time–room slots [24].
2. Cluster methods, in which the problem is divided into a number of event sets. Each set is defined so that it satisfies all hard constraints. Then, the sets are assigned to real time–room slots to satisfy the soft constraints too [28].
3. Constraint-based methods, according to which a timetabling problem is modeled as a set of variables (events) to which values (resources such as teachers and rooms) have to be assigned in order to satisfy a number of constraints [9, 16].
4. Meta-heuristic methods, such as genetic algorithms, simulated annealing, tabu search, and other heuristic approaches, that are mostly inspired from nature, and apply nature-like processes to solutions or populations of solutions, in order to evolve them towards optimality [2, 11, 19, 23, 26, 27].

When applying genetic algorithms to some optimization or scheduling problem, the crucial element is encoding. For a timetabling problem there are two main types of encoding: direct [23] and indirect [19]. The advantage of direct encoding is that the whole search space can be encoded, but it usually leads to violation of many hard constraints. Indirect encoding is based on some rules or instructions for building the resulting timetable and so there is less probability

of hard constraint violation, but it can reach only a limited portion of the search space and thus it can be more easily trapped in a local optimum. In our application a combination of both encoding types was used, because we want to let the algorithm decide itself whether to use the direct or indirect representation, and the ratio of using both of them. The timetable builder is based on the order of lectures and time–room slots encoded in the chromosome. By this approach all the feasible timetables can be addressed and the probability of generating an infeasible timetable is strongly reduced. Finally the proposed self-adaptive genetic algorithm is compared to standard genetic algorithms on this timetabling problem. Also the role of enrollment optimization algorithm is discussed.

## 2 The Timetabling Problem

In this section we describe the input data for the university timetabling problem and formalize the optimization model. In the model we use the following notation:

$n_R$	number of available rooms $R_1, R_2, \dots, R_{n_R}$
$n_U$	number of subjects $U_1, U_2, \dots, U_{n_U}$
$n_L$	number of lectures (events) $L_1, L_2, \dots, L_{n_L}$
$n_S$	number of students $S_1, S_2, \dots, S_{n_S}$
$n_T$	number of teachers $T_1, T_2, \dots, T_{n_T}$
$n_M$	number of time slots $M_1, M_2, \dots, M_{n_M}$
$n_G$	number of time–room slots $G_1, G_2, \dots, G_{n_G}$
$C$	clash matrix with elements $c_{ij}; i = 1, 2, \dots, n_L; j = 1, 2, \dots, n_L$
$P$	preference matrix with elements $p_{ij}; i = 1, 2, \dots, n_L; j = 1, 2, \dots, n_G$ .

The purpose of the clash matrix  $C$  is to determine which lectures should not be scheduled at the same time. Each element of the clash matrix  $c_{ij}$  is equal to the number of students which are enrolled to both lectures  $L_i$  and  $L_j$ . The number of students that attend more lectures at the same time is only a soft constraint, because each student has an individual timetable and so it is nearly impossible to build a timetable with no clashes for students. The clash matrix  $C$  is also used for handling teachers clashes, i.e. the high penalty coefficient is set to the matrix element  $c_{ij} = 10^6$  for all lectures  $L_i$  and  $L_j$  which are taught by the same teacher.

The purpose of the preference matrix  $P$  is to set preference  $p_{ij}$  to particular time–room slot  $G_j$  for each lecture  $L_i$ . Note that the preferences are taken as negative preferences, i.e. the higher the element of the preference matrix  $p_{ij}$ , the less suitable is the time–room slot  $G_j$  for the lecture  $L_i$ . The matrix  $P$  is also used for handling the suitable rooms for each lecture. By the matrix  $P$  we can handle both the teacher time preferences and the suitable rooms for each lecture. The teacher time preferences are usually taken as soft constraints, i.e. only a small penalty coefficient is set to  $p_{ij}$  for all time–room slots  $G_j$  that correspond to less preferred timeslots for the lectures  $L_i$  that are taught by the teacher. The requirement for suitable room is handled as a hard constraint, so

the high penalty coefficient  $p_{ij} = 10^6$  is set for all time–room slots  $G_j$  that correspond to unsuitable rooms for particular lecture  $L_i$ .

The core of the timetabling problem is to assign suitable timeslot  $G_j$  to each lecture  $L_i$  such that all hard constraints are satisfied and the number of soft constraint violations was minimal. This problem can be mathematically formulated as an optimization model minimizing error of the timetable defined as

$$z = \sum_{i=1}^{n_L} \left[ \sum_{j=1}^{n_G} x_{ij} p_{ij} + \sum_{k=1}^{n_L} c_{ik} \text{sametime}(i, k) \right] + w_p \cdot \text{penalty}(x_{ij}) \rightarrow \min \quad (1)$$

$$\text{s.t. } \sum_{i=1}^{n_L} x_{ij} \leq 1 \text{ for } j = 1, 2, \dots, n_G$$

$$\sum_{j=1}^{n_G} x_{ij} = 1 \text{ for } i = 1, 2, \dots, n_L,$$

where  $x_{ij}$  is a binary optimized variable determining whether the lecture  $L_i$  is taught in the time–room slot  $G_j$ . The expression  $\text{sametime}(i, k)$  is the function that is equal to 1 if the lecture  $L_i$  is taught at the same time as the lecture  $L_k$ , otherwise it is equal to zero. The expression  $\text{penalty}(x_{ij})$  is the function determining the penalty of the timetable that it is not possible to express by clash matrix  $C$  or preference matrix  $P$ . The coefficient  $w_p$  is the weight of penalty( $x_{ij}$ ) by which it contributes to the error of the timetable.

### 3 Teacher Preferences

To express teacher preferences we define the following criteria. For each criterion the teacher sets its preference  $t^*$  by ‘mark’ from 1 to 5; mark 1 means that it is the best and mark 5 means that it is the worst. Because the preferences  $t^*$  are used for calculation of total error of the timetable, and  $t^* = 1$  means the best possibility for the teacher and actually no error of timetable, we transform  $t^*$  to  $t^{*’}$  by decreasing by 1, i.e.  $t^{*’} = t^* - 1$ . If the preference  $t^* = 5$  it is a hard constraint, so  $t^{*’} = 10^6$ .

#### 3.1 Time Preferences

For each timeslot  $M_k$  the teacher must set its preference  $t_k^M$ , for which we calculate  $t_k^{M’} = t_k^M - 1$ . After that we assign  $p_{ij} = w_M \cdot t_k^{M’}$  for all lectures  $L_i$  that are taught by this teacher and for all timeroom slots  $G_j$  that corresponds to timeslot  $M_k$ . The parameter  $w_M$  is a weight by which it contributes to the error of the timetable. For example if the teacher cannot teach on Wednesdays, on Mondays and Tuesdays he prefers to teach in the afternoon and on Thursdays and Fridays he prefers to teach in the morning, then the preferences  $t_k^M$  are as follows:

	08:00– 08:45	08:50– 09:35	09:40– 10:25	10:30– 11:15	11:20– 12:05	12:10– 12:55	13:00– 13:45	13:50– 14:35	14:40– 15:25	15:30– 16:15
Mon	4	3	3	3	2	2	1	1	1	1
Tue	4	3	3	3	2	2	1	1	1	1
Wed	5	5	5	5	5	5	5	5	5	5
Thu	1	1	1	1	1	2	2	3	3	3
Fri	1	1	1	1	1	2	2	3	3	3

### 3.2 Number of Teaching Days per Week

For each number of days the teacher must set  $t_k^N$ , for which we calculate  $t_k^{N'} = t_k^N - 1$ . We calculate the number of days  $d$  in which the teacher teaches at least one lecture and then increase the value of  $\text{penalty}(x_{ij})$  by the value of  $w_N \cdot t_d^{N'}$ , where  $w_N$  is the weight by which it contributes to the penalty of timetable. For example, if the teacher would like to teach 2 or 3 days a week, 4 days it is not convenient for him, 5 days is not acceptable for him and in 1 day it is not possible to teach all lectures, then the preferences  $t_k^N$  look like

Number of teaching days	1	2	3	4	5
Preferences $t_k^N$	5	1	1	3	5

### 3.3 Length of Teaching Block Without Break

By this criterion the teacher sets whether he prefers to concentrate lectures in one long teaching block or to disperse them to several short teaching blocks. For each length of the teaching block (in hours) the teacher must set its preference  $t_k^B$ , for which we calculate  $t_k^{B'} = t_k^B - 1$ . We calculate for each continuous teaching block its length  $l$  and then increase the value of  $\text{penalty}(x_{ij})$  by the value of  $w_B \cdot t_l^{B'}$ , where  $w_B$  is the weight by which it contributes to the penalty of timetable. For example, if the teacher does not like to have too dispersed lectures, i.e. he wants to teach at least 2 hours without break, preferably he would like to teach 3–4 continuous hours, 5 continuous hours is very exhausting and more than 5 continuous hours is not possible to teach, then the preferences  $t_k^B$  look like

Length of block	1	2	3	4	5	6	7	8	9	10
Preferences $t_k^B$	5	3	1	1	3	5	5	5	5	5

### 3.4 Number of Teaching Hours per Day

For each number of the teaching hours the teacher must set its preference  $t_k^H$ , for which we calculate  $t_k^{H'} = t_k^H - 1$ . We calculate for each day the number of teaching hours  $h$  and then increase the value of  $\text{penalty}(x_{ij})$  by the value of  $w_H \cdot t_h^{H'}$ , where  $w_H$  is the weight by which it contributes to the penalty of



timetable. For example, if it is very inconvenient for the teacher to go to school to teach only 1 or 2 hours, optimal number is 3–5 hours per day, 6–7 hours is exhausting and above 7 hours per day is impossible, then the preferences  $t_k^H$  look like

Number of hours	1	2	3	4	5	6	7	8	9	10
Preferences $t_k^H$	5	4	2	1	1	3	4	5	5	5

### 3.5 Span of Teaching Day

This criterion means the difference between beginning of the first lecture and the end of last lecture in a day, i.e. the sum of teaching hours and breaks between them. For each length of span the teacher must set its preference  $t_k^S$ , for which we calculate  $t_k^{S'} = t_k^S - 1$ . We calculate for each teaching day the length of span  $s$  and then increase the value of penalty( $x_{ij}$ ) by the value of  $w_S \cdot t_s^{S'}$ , where  $w_S$  is the weight by which it contributes to the penalty of timetable.

### 3.6 Length of Continuous Break

By this criterion the teacher sets how many hours he needs to relax. For each number of relax hours the teacher must set its preference  $t_k^R$ , for which we calculate  $t_k^{R'} = t_k^R - 1$ . We calculate for each break between two teaching blocks its length  $r$  and then increase the value of penalty( $x_{ij}$ ) by the value of  $w_R \cdot t_r^{R'}$ , where  $w_R$  is the weight by which it contributes to the penalty of timetable. Of course if necessary it is possible to incorporate other teacher preferences in similar way as previous ones.

## 4 Enrollment of Students

At most universities there are some groups of students which share the same timetable. But at some universities including Silesian University each student has an individual timetable, i.e. there are no groups of students which have the same timetable, and it is even hard to find any two students that have the same timetable. Thus solving the problem becomes very complex.

At Silesian University each student can choose subjects that he wants to study. If the subject consists of only one lecture there is usually no problem as the student is automatically enrolled to that lecture. But most subjects consist of two kinds of meetings: classical lectures and seminars. There are usually more seminars of the same subject, but the student can be enrolled only to one of them. The question is how to set an appropriate seminar for each student. One possibility is to do it randomly, but by this way it will be very difficult or

nearly impossible to build a timetable in which each student has an unclashing timetable or the number of clashing lectures for students is acceptably small. So we propose an algorithm for optimized enrollment of students that minimizes the number of clashing constraints for students.

In the model we use the following notation:

- $S_{ij}^U$  binary variable defining whether student  $S_i$  is enrolled to subject  $U_j$
- $S_{ij}^L$  binary variable defining whether student  $S_i$  is enrolled to lecture  $L_j$
- $U_{ij}^L$  binary variable defining whether subject  $U_i$  contains the lecture  $L_j$
- $L_i^S$  maximal number of students that can be enrolled to lecture  $L_i$ .

Without loss of generality suppose that each kind of lecture of the same subject will be labeled as a different subject. First we set the elements  $c_{ij}^T$  of clash matrix  $C$  for teacher clashes as described in Section 2. After that, students are enrolled to the lectures corresponding to the subjects which have only one lecture of the same type, i.e. there is no possibility of choice of lecture to which a student should be enrolled. The core of the enrollment problem is then to enroll all students to all lectures such that all constraints are satisfied and the number of nonzero elements  $c_{ij}$  of the clash matrix  $C$  is minimal. The problem can be mathematically formulated as an optimization model minimizing the number of nonzero elements  $c_{ij}$  defined as

$$c = \sum_{i=1}^{n_L} \sum_{j=1}^{n_L} \text{nonzero}(c_{ij}) \rightarrow \min \tag{2}$$

$$\text{s.t.} \quad \sum_{i=1}^{n_S} S_{ij}^L \leq L_j^S \quad \text{for } j = 1, 2, \dots, n_L$$

$$\sum_{j=1}^{n_U} S_{ij}^U = \sum_{k=1}^{n_L} S_{ik}^L \quad \text{for } i = 1, 2, \dots, n_S$$

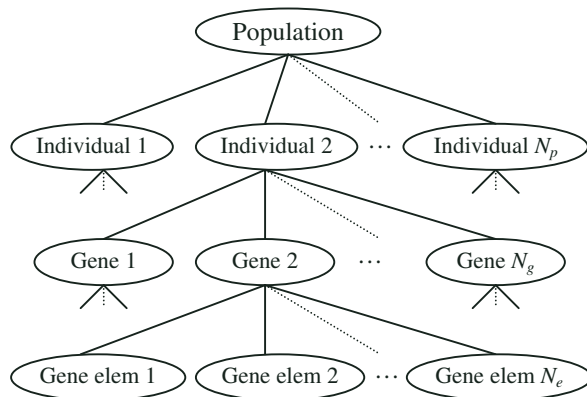
$$\sum_{j=1}^{n_L} S_{ij}^L \cdot U_{kj}^L \cdot S_{ik}^U = 1 \quad \text{for } i = 1, 2, \dots, n_S, k = 1, 2, \dots, n_U$$

where  $c_{ij} = c_{ij}^T + \sum_{k=1}^{n_S} S_{ki}^L \cdot S_{kj}^L$  for  $i, j = 1, 2, \dots, n_L$  and  $\text{nonzero}(c_{ij})$  is a function which is equal to 0 when  $c_{ij}$  is zero and 1 otherwise.

## 5 Self-adaptive Genetic Algorithm

As the proposed timetabling problem is very complex, a genetic algorithm with self-adaptation is used for its solving. In this section the basic characteristic of the algorithm will be described.

Encoding is a crucial element of every genetic algorithm. The structure of our self-adaptive genetic algorithm's encoding is depicted in Figure 1. The idea



**Fig. 1.** The structure of a population

behind the proposed encoding consists in redundancy of information through hierarchical the evaluation of individuals.

As we can see, in the population each individual is composed of  $N_g$  genes, where each gene corresponds to exactly one optimized variable. Each gene is composed of  $N_e$  gene elements. The number of gene element is different for each gene and it varies through evolution. Each gene element contains low-level parameters, which encode optimized variables and parameters of evolution. All parameters are listed in Table [1](#).

The upper index ‘ $E$ ’ denotes that it is a gene element value of the parameter. As the encoding is hierarchical, there are several levels of the parameters, so gene values of parameters are marked by the upper index ‘ $G$ ’, individual values by ‘ $I$ ’ and population values by ‘ $P$ ’.

Since genetic operators are applied only to the low level values of parameters (gene element), the upper level values of parameters cannot be altered directly through the evolution process, but only indirectly by an evaluation mechanism from low-level values.

### 5.1 Mechanism of Gene Evaluation

The proposed encoding is polyploiditital, so each gene is composed of  $N_e$  gene elements. The number of gene elements is variable and undergoes evolution. For evaluation of gene values of gene elements we use the simple arithmetical average, i.e.

$$X^G = \frac{1}{N_e} \sum_{i=1}^{N_e} X_i^E, \quad (3)$$

where  $X$  stands for parameters that must be evaluated, i.e.  $x, s_m, s_w, r_r, r_t, r_p, c_d, N_p, i_t$ .

**Table 1.** The structure of a gene element

Name	Description	Range
$x^E$	Optimized variable	$\langle 0; 1 \rangle$
$q_m^E$	Parameter of mutation	$\langle -1; 1 \rangle$
$q_p^E$	Parameter of protected mutation	$\langle -1; 1 \rangle$
$r_m^E$	Radius of mutation	$\langle 0; 0.5 \rangle$
$p_c^E$	Probability of crossover	$\langle 0; 1 \rangle$
$r_c^E$	Ratio of crossover	$\langle 0; 1 \rangle$
$q_d^E$	Parameter of deletion	$\langle -0.1; 0.1 \rangle$
$q_u^E$	Parameter of duplication	$\langle -0.1; 0.1 \rangle$
$q_t^E$	Parameter of translocation	$\langle -0.1; 0.1 \rangle$
$s_m^E$	Identifier of myself for mating	$\langle 0; 1 \rangle$
$s_w^E$	Wanted partner for mating	$\langle 0; 1 \rangle$
$r_r^E$	Ratio of replacement	$\langle 0; 1 \rangle$
$r_t^E$	Ratio of population for selection	$\langle 0; 1 \rangle$
$r_p^E$	Ratio of population for 2nd partner selection	$\langle 0; 1 \rangle$
$c_d^E$	Coefficient of death	$\langle 0; 1 \rangle$
$N_p^E$	Wanted size of population	$\langle 0; 1 \rangle$
$i_t^E$	Identifier of translocation	$\langle 0; 1 \rangle$

### 5.2 Mechanism of Individual Evaluation

Parameters concerning the whole individual, such as  $s_m^I, s_w^I, r_r^I, r_t^I, r_p^I, c_d^I, N_p^I$  are evaluated as simple arithmetical average, i.e.

$$X^I = \frac{1}{N_g} \sum_{i=1}^{N_g} X_i^G. \tag{4}$$

The number of genes  $N_g$  is not variable, because one gene contains exactly one optimized variable.

### 5.3 Mechanism of Population Evaluation

Parameters concerning the whole population, such as  $r_r^P, r_t^P, c_d^P, N_p^P$ , are evaluated as weighted average with weights according to their relative fitness  $w_f$ , defined as

$$w_f = \frac{N_p^P - i + 1}{\frac{(1+N_p^P)N_p^P}{2}} \tag{5}$$

where  $i$  is index of  $i$ th individual in population sorted by fitness in descending order, i.e. the individual with the highest value of the fitness function has the value of  $i$  equal to 1, the individual with the second highest value of the fitness function has the value of  $i$  equal to 2, etc.

## 6 Genetic Operators

As the proposed encoding is specific, the genetic operators must be adjusted to fit the encoding. There are used not only common genetic operators as selection, crossover or mutation, but also some specific ones, as described in the following paragraphs. Of course the genetic algorithms are powerful enough to work only with the basic genetic operators, but the idea is to use as many operators as possible and let the genetic algorithm decide which ones to use through self-adaptation.

### 6.1 Selection

In genetic algorithms the selection of both parents for mating is usually based on their fitness, but this is not true in nature. In nature a winner of a tournament selects his partner according to his individual preferences. Importantly, he cannot take into account his genotype directly, i.e. the values of his genes or his fitness, but only his phenotype, i.e. only the expression of the genes to the outside. In a similar way we try to imitate nature by using parameters  $s_m^I$  and  $s_w^I$ . The parameter  $s_w^I$  represents an individual's preferences for mating and the parameter  $s_m^I$  represents an individual's phenotype for mating. So the first parent is selected by a tournament selection method with variable ratio of population  $r_t^P$  from which the fittest individual is selected. The second parent is selected according to the individual's preferences represented by the parameter  $s_w^I$ , i.e. the first parent selects an individual with the minimal value of expression  $|s_w^I - s_m^I|$ , but this selection is made from only a limited ratio of population  $r_p^I$ .

### 6.2 Crossover

The crossover operator is applied to every gene element of the first parent with the probability  $p_c^E$ . The crossover itself proceeds only between gene elements of mating parents according to

$$X_3^E = X_1^E + (X_2^E - X_1^E) \cdot r_c^E \quad (6)$$

where  $X$  stands for all parameters of a gene element (see Table 1),  $r_c^E$  is a ratio of crossover of the first parent defined in this gene element, the lower index '1' denotes the gene element of the first parent, the index '2' the second parent and the index '3' denotes the child of both parents. The gene element of the second parent is selected randomly, but it is of the same gene as the gene element of the first parent.

### 6.3 Mutation

The mutation operator is applied to every gene element with probability  $p_m^E = |q_m^E|$ . Notice that probability of mutation is calculated as the absolute value of the parameter of mutation  $q_m^E \in \langle -1; 1 \rangle$ , because the mean value of  $p_m^E$  should

be zero. Moreover, every gene element has its own probability of mutation. The mutation formula is defined as

$$X_{\text{new}}^E = X_{\text{old}}^E + (X_{\text{max}}^E - X_{\text{min}}^E) \cdot U(-r_m^E, r_m^E) \quad (7)$$

where  $X$  stands for all parameters of the gene element,  $U(a, b)$  is a random variable with uniform probability distribution in the interval  $\langle a; b \rangle$ ,  $X_{\text{new}}^E$  is the value of the parameter after mutation,  $X_{\text{old}}^E$  is the original value of the parameter, and  $X_{\text{max}}^E$  ( $X_{\text{min}}^E$ ) is the maximal (minimal) allowed bit element value of the parameter as defined in Table 1.

#### 6.4 Duplication

The duplication operator is applied to every gene element with probability  $p_u^E = |q_u^E|$ . The gene element is duplicated (copied) with the same value of all parameters with the only exception being that the values of parameter  $q_u^E$  of both gene elements are divided by 2, in order to inhibit exponential growth of the number of bit elements.

#### 6.5 Deletion

The deletion operator is applied to every gene element with probability  $p_d^E = |q_d^E|$ . This means that the gene element is simply removed from the particular gene. The degree of polyploidy is controlled by deletion and duplication operators.

#### 6.6 Translocation

Translocation means that a gene element is moved from its original gene to one of neighboring genes with probability  $p_t^E = |q_t^E|$ . However, the neighboring gene may decide whether to accept the gene element: that is, the real probability of translocation is defined as

$$p = p_t^E \cdot \left(1 - \left| i_t^{G(\text{new})} - i_t^{G(\text{old})} \right| \right) \quad (8)$$

where  $i_t^{G(\text{new})}$  is the identifier of translocation of the gene to which the gene element is going to move, and index '(old)' denotes the original gene. The values of the gene element's parameters are left unchanged with the only exception that  $q_t^E$  is multiplied by coefficient  $-0.5$ , in order to decrease further translocations. The gene element decides whether to translocate to the left or right neighboring gene according to the sign of  $q_t^E$ .

#### 6.7 Protected Mutation

Protected mutation is an analogy of local optimization and it is applied only to the fittest individual in the population after application of all previous operators and after values of fitness function of all individuals in the population have been calculated. The protected mutation operator is applied to every gene element

with probability  $p_p^E = |q_p^E|$  and after that the new value of fitness function is calculated and compared to the value of fitness function before applying the protected mutation operator. If the new value of fitness function is greater than previously, then the mutated chromosome is used, otherwise the old chromosome is used for the following evolution cycle.

## 6.8 Replacement of Individuals

For every individual the parameter of life strength,  $L$ , is defined. When the individual is created its life strength  $L$  is set to one and in every generation it is multiplied by the coefficient  $c_L$  defined as

$$c_L = 1 - c_d^P (1 - w_f). \quad (9)$$

Evidently, through evolution, a less fitter individual causes a greater decrease in  $L$ .

In every generation all  $X^P$  parameters are evaluated and by using the above listed genetic operators  $N_p^P \cdot r_r^P$  new individuals are created. Then a randomly selected individual is killed with probability  $(1 - L)$ . This process of killing individuals is repeated until only  $N_p^P$  individuals survive in the population.

## 7 Mapping the Timetabling Problem to the Chromosome

The timetabling problem actually consists of two tasks. In the first one students must be enrolled to lectures and in the second one there must be lectures assigned to time–room slots. In this section the process of decoding from the chromosome will be described.

### 7.1 The Enrollment Builder

First we enroll students to lectures according their preferences for subjects they want to attend by the process described in Section 4. For optimization of enrollment of students the proposed self-adaptive genetic algorithm was used. As was mentioned in Section 5, each gene of a chromosome represents one real variable within the interval  $\langle 0; 1 \rangle$ . In order to apply this chromosome encoding for the enrollment problem, the chromosome is divided into two parts. The first part  $A$  consisting of  $(n_S \cdot n_U)$  genes represents the parameters for all subjects selected by students and the second part  $B$  consisting of  $n_L$  genes represents parameters for lectures. The main idea behind the encoding of the lecture enrollment is that the subjects selected by students are sorted in ascending order according to values of parameters in the part  $A$  of the chromosome and then in this order the lecture enrollment builder assigns the first free suitable lecture with the least difference of  $|A_i - B_j|$ , where  $A_i$  is the  $i$ th parameter of part  $A$  of the chromosome and  $B_j$  is the  $j$ th parameter of part  $B$  of the chromosome. The fitness function  $f$  for the genetic algorithm is the negative value of  $c$  in (2), i.e.  $c = -z$ .

### 7.2 The Timetable Builder

For solving the university timetabling problem the self-adaptive genetic algorithm was used, too. The process of encoding is similar to encoding of the enrollment problem above. The chromosome is divided into three parts. The first part  $A$ , consisting of  $n_L$  genes, represents the parameters for lectures; the second part  $B$ , consisting of  $n_G$  genes, represents parameters for time–room slots; and the last part contains control parameters for the timetable builder. Lectures are sorted in ascending order according to the values of parameters in part  $A$  of the chromosome and then in this order the timetable builder assigns the first suitable unused time–room slot with the least difference of  $|A_i - B_j|$ , where  $A_i$  is the  $i$ th parameter of part  $A$  of the chromosome and  $B_j$  is the  $j$ th parameter of part  $B$  of the chromosome. Whether the time–room slot  $G_j$  is suitable for the lecture  $L_i$  is determined by the control parameter  $D$  in the last part of the chromosome. The parameter  $D$  contains the maximal accepted penalty of assigning lecture  $L_i$  to time–room slot  $G_j$ , which is calculated by formula (1). If there is no suitable time–room slot for the lecture  $L_i$ , the best suited still unused time–room slot is selected for the lecture. The fitness function  $f$  for the genetic algorithm is the negative value of  $z$  in (1), i.e.  $f = -z$ .

To make the idea behind decoding the chromosome clearer, a simple example will be provided. Let us suppose we have three lectures:  $L_1, L_2, L_3$  and four time–room slots:  $G_1, G_2, G_3, G_4$ . So the chromosome for such simple timetable will have eight genes. Let us suppose that after evaluation, the gene values of variables  $x^G$  are

Part	A			B				D
Description	$L_1$	$L_2$	$L_3$	$G_1$	$G_2$	$G_3$	$G_4$	
Value	0.45	0.91	0.39	0.82	0.36	0.49	0.56	0.8

First lectures must be sorted according values of  $x^G$ , so the order will be  $L_3, L_1, L_2$ . For all lectures we now must calculate difference between the lecture and particular time–room slot  $|A_i - B_j|$ :

	$G1$	$G2$	$G3$	$G4$
$L3$	0.43	<b>0.03</b>	0.10	0.17
$L1$	0.37	<i>0.09</i>	<b>0.04</b>	0.11
$L2$	<b>0.09</b>	<i>0.55</i>	<i>0.42</i>	0.35

The selected time–room slot with least difference of  $|A_i - B_j|$  for each lecture is marked by bold font and time–room slots used for previous lectures are in italic. So the resulting timetable according chromosome provided in the example will be  $L1 \rightarrow G3, L2 \rightarrow G1, L3 \rightarrow G2$ .



## 8 Numerical Experiments

This model was then applied for solving the real timetabling problem in the School of Business Administration at Silesian University. The problem size and its structure can be characterized by the values of parameters: number of rooms  $n_R = 43$ , number of subjects  $n_U = 340$ , number of lectures  $n_L = 705$ , number of students  $n_S = 1807$ , number of teachers  $n_T = 112$ , number of time slots  $n_M = 60$ , number of time-room slots  $n_G = 2400$ . When evaluating the error of timetable  $z$  defined in (1), we must set up weights of the criteria:  $w_M = 3$ ,  $w_N = 5$ ,  $w_B = 3$ ,  $w_H = 3$ ,  $w_S = 2$ ,  $w_R = 2$ ,  $w_P = 0.5$ . The number of computers that were used was  $N_a = 30$ .

The best solution found by the self-adaptive genetic algorithm was the timetable with the minimal value of error function  $z = 7184$ . The resulting timetable satisfied all hard constraints and there were 83 students that had any clashing lecture. The previously used approach for constructing the timetable produced a timetable in which there were on average 2.8 clashing lectures for each student, moreover it was a very boring and time-consuming process, because the timetable was made manually, with the computer used only as a graphical user interface.

In order to also test the performance of the proposed self-adaptive genetic algorithm (SAGA) we have compared it with the simple genetic algorithm (SGA) on this timetabling problem. The simple genetic algorithm used a binary encoding, the size of population was 30 individuals, probability of mutation 0.003 and elitism was used. Maximal number of generations for both algorithms was  $10^4$ . We ran both algorithms 10 times and measured the average penalty function  $z$  of the best timetable found in each run of both genetic algorithms. The average best value of error function for SAGA was 7331 and for SGA the average value of  $z$  was 7687. As can be seen, SAGA was slightly better, but the main advantage of SAGA is that there is no need for finding values of the parameters, as there are no parameters set in advance. The average CPU time on the processor Pentium 1.7GHz was about 8 hours for the SAGA and about 7 hours for the SGA for single run of 10 000 generations.

We also tested the role of the enrollment optimization algorithm. The enrollment optimization algorithm as described in Section 4 was substituted by random enrollment of students to lectures and the best solution found by the SAGA was the timetable with the minimal value of error function  $z = 12553$ . As can be seen, it is much worse than with applying the enrollment optimization algorithm.

## 9 Conclusions

In this paper we have designed an optimization model for solving the university timetabling problem that is capable of dealing with individual timetables of every student. For solving the timetabling problem we have proposed a self-adaptive genetic algorithm with self-adaptation of all its parameters. This algorithm was applied for solving the real university timetabling problem at Silesian University.

It was shown that the self-adaptive genetic algorithm is able to effectively solve the timetabling problem. It was also shown how to significantly decrease the number of student clash constraints by the proposed enrollment optimization algorithm when dealing with individual timetables of students.

A great problem appeared when it was applied to the real timetabling problem with changed preferences and requirements for timetable, because the new timetable is completely different to the original one. So further study will be concerned to deal with the problem of minimization of the number of changes between the new and original timetable. For the self-adaptive genetic algorithm it would be beneficial to test the importance of every genetic operator and also to test the effectiveness of the algorithm on other combinatorial problems.

## References

1. Abdullah, S., Burke, E.K., McCollum, B.: An investigation of variable neighborhood search for university course timetabling. In: MISTA. Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications, New York, pp. 413–427 (July 2005)
2. Abramson, D.: Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science* 37, 98–113 (1991)
3. Aickelin, B.E.K., Li, J.: Improved squeaky wheel optimisation for driver scheduling. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) *Parallel Problem Solving from Nature - PPSN IX*. LNCS, vol. 4193, pp. 182–191. Springer, Heidelberg (2006)
4. Bacardit, J., Krasnogor, N.: Smart crossover operator with multiple parents for a Pittsburgh learning classifier system. In: GECCO 2006. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 1441–1448. ACM Press, New York (2006)
5. Bäck, T.: Self-adaptation in genetic algorithms. In: *Proceedings of the 1st European Conference on Artificial Life*, MIT Press, Cambridge, MA (1992)
6. Beyrouthy, C., Burke, E.K., Landa-Silva, D., McCullom, B., McMullan, P., Parkes, A.J.: The teaching space allocation problem with splitting. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006*. LNCS, vol. 3867, pp. 232–252. Springer, Heidelberg (2007)
7. Bufé, M., Fischer, T., Gubbels, H., Häcker, C., Hasprich, O., Scheibel, C., Karsten Wecker, K., Wecker, N., Wenig, M., Wolfangel, C.: Automated solution of a highly constrained school timetabling problem – preliminary results. In: *Proceedings of the Evo Workshops*, Como, Italy, Springer, Berlin (2001)
8. Burke, E.K., Newall, J.: Enhancing timetable solutions with local search methods. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 195–206. Springer, Heidelberg (2003)
9. Brailsford, S.C., Potts, C.N., Smith, B.M.: Constraint satisfaction problems: algorithms and applications. *European Journal of Operational Research* 119, 557–581 (1999)
10. De Jong, K.A.: An analysis of the behavior of a class of genetic adaptive systems. Ph.D. Thesis, University of Michigan (1975)
11. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001)

12. Even, S., Iati, A., Shamir, A.: On the complexity of timetabling and multicommodity flow problems. *SIAM Journal of Computation* 5, 691–703 (1976)
13. Fernandes, C., Caldeira, J.P., Melicio, F., Rosa, A.: High school weekly timetabling by evolutionary algorithms. In: *Proceedings of the 14th Annual ACM Symposium on Applied Computing*, San Antonio, TX (1999)
14. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA (1989)
15. Kendall, G., Mohd, H.N.: Tabu search hyper-heuristic approach to the examination timetabling problem at University Technology MARA. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 199–217. Springer, Heidelberg (2005)
16. Legierski, W.: Search strategy for constraint-based class–teacher timetabling. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 247–261. Springer, Heidelberg (2003)
17. Marsili, S.L., Alba, P.A.: Adaptive mutation in genetic algorithms. *Soft Computing* 4, 76–80 (2000)
18. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edn. Springer, New York (1996)
19. Paechter, B., Rankin, R.C., Cumming, A., Fogarty, T.C.: Timetabling the classes of an entire university with an evolutionary algorithm. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) *Parallel Problem Solving from Nature - PPSN V*. LNCS, vol. 1498, pp. 865–874. Springer, Heidelberg (1998)
20. Perzina, R.: Self-adaptation in genetic algorithms. In: *SCI 2003. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, IIIS, Orlando, FL, pp. 234–238 (2003)
21. Perzina, R.: A self-adapting genetic algorithm for solving the university timetabling problem. In: *SCI 2004. Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics*, IIIS, Orlando, FL, pp. 284–288 (2004)
22. Perzina, R., Ramík, J.: A new portfolio selection model solved by genetic algorithms. In: *Proceedings of the 20th International Conference of MME*, VŠB TU, Ostrava, pp. 201–207 (2002)
23. Ross, P., Hart, E., Corne, D.: Some observations about ga-based exam timetabling. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 115–129. Springer, Heidelberg (1998)
24. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
25. Stefano, C.D., Tettamanazi, A.G.B.: An evolutionary algorithm for solving the school timetabling problem. In: *Proceedings of the Evo Workshops 2001*, Como, Italy, pp. 452–462. Springer, Berlin (2001)
26. Terashima-Marin, H., Ross, P., Valenzuela-Rendon, M.: Evolution of constraint satisfaction strategies in examination timetabling. In: *GECCO 1999. Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 635–642. Morgan Kaufmann, San Mateo, CA (1999)
27. Thompson, J.M., Dowsland, K.A.: A robust simulated annealing based examination timetabling system. In: *Computers and Operations Research*, vol. 25, pp. 637–648. Elsevier, Oxford (1998)
28. White, G.M., Chan, P.W.: Towards the construction of optimal examination timetables. *INFOR* 17, 219–229 (1979)
29. Yang, J.M., Kao, C.Y.: Integrating adaptive mutations and family competition into genetic algorithms as function optimizer. *Soft Computing* 4, 89–102 (2000)

# **School Timetabling**

# A Case Study for Timetabling in a Dutch Secondary School

Peter de Haan<sup>1</sup>, Ronald Landman<sup>2</sup>, Gerhard Post<sup>1,3</sup>, and Henri Ruizenaar<sup>3,4</sup>

<sup>1</sup> ORTEC, Groningenweg 6k, 2803 PV Gouda, The Netherlands

<sup>2</sup> KLM Royal Dutch Airlines, Amsterdamseweg 55,  
1182 GP Amstelveen, The Netherlands

<sup>3</sup> Department of Applied Mathematics, University Twente,  
PO Box 217, 7500 AE Enschede, The Netherlands

<sup>4</sup> Stedelijk Lyceum, Locatie Kottenpark, Lyceumlaan 30,  
7522 GK Enschede, The Netherlands

**Abstract.** This paper describes a case study for constructing the yearly schedule of a secondary school in the Netherlands. This construction is divided in three steps. In the first step we create cluster schemes containing the optional subjects. A cluster scheme consists of cluster lines, and a cluster line contains classes which will be taught simultaneously. Part of the problem is that the students are not yet assigned to the classes. Once the cluster schemes are fixed, it remains to schedule the lessons to time slots and rooms. We first schedule the lessons to day-parts, and once this is completed we schedule the lessons to time slots within the day-parts. Thanks to consistency checks in the day-part phase, going from day-parts to time slots is possible. Finally, in the third step, we improve the previously found schedule by a tabu search using ejection chains. Compared to hand-made schedules, the results are very promising.

## 1 Introduction

In the past 25 years a lot of research has been done on automated High School Timetabling. This research can be divided in two groups:

1. Theoretical oriented research and surveys, see for example, in chronological order [5,7,9,18,21,23,24]. These papers either define some concepts and/or methods, but do not describe real-life implementations.
2. Research based on several cases (usually high schools from the region). These papers (hopefully) define the problem they study, and explain that their methods perform quite well on the real-life cases considered. Examples of these papers are found below.

What is apparent from the studies in the second class, is that the problems differ widely among the countries. Of course, there are certain aspects that they all have in common. This could be named the basic high school timetabling feasibility problem replacing ‘lesson’ by ‘event’, this is the basic timetabling problem):

*Given a set of lessons with needed resources, and time slots,*

*Assign resources and a time slot to each lesson, such that the resources are not over-used.*

We assume the situation that we need to construct a schedule for a week, which is repeated for a certain period like a year or semester. A lesson has usually the following resources with restrictions:

1. *Class*: the (virtual) group; can be used once per time slot. There are two principally different situations:
  - The classes are mutually disjoint: every student is in exactly one class. This is called the ‘Class–Teacher model’.
  - The classes are not disjoint: the class depends on the subject (students have optional subjects). This occurs for example in Germany [4,11,25], the Netherlands [12,26], and New Zealand [27].
 In the first case, some intermediate cases can exist, where several classes are combined and reshuffled based on level (easy math vs. difficult math), or religion, or sex (physical education lessons).
2. *Subject*: the subject of the lesson; the (subject, class) combination can be used once per day.
3. *Students*: the students that constitute the class of the lesson; a student can be used once per time slot. In most countries the students are preassigned to the class. However in case of optional subjects, these students might have to be divided over different classes (the Netherlands: [12,20,26]).
4. *Teacher(s)*: the teachers of the lesson; a teacher can be used once per time-slot, if the teacher is available at that time. Usually a lesson has just one teacher, which in most countries seems to be preassigned, while in some it has to be assigned, for example in Australia [1], Greece [6], and the UK [28].
5. *Room(s)*: the rooms needed for the lesson; a room can be used once per time slot. Usually the lesson needs only one room, and this room has to be assigned; some papers mention that a class has its own room (Greece [22], Italy [19]). (Another possibility is that only room types – like music-room, gymnasium, etc. – are assigned.)

From this we see that already the basic feasibility problem has several variants: assign students or not, assign teachers or not, assign rooms or not.

As far as the objective function is concerned, the variants are even more diverse. Here we mention two cases, which sometimes appear as hard constraints:

1. Compact schedules for classes, which means schedules for classes without idle times. Here an idle time (for a class or teacher) is defined as a free time slot between the first and last lesson of the day. In several cases this is automatic, as a class has as many lessons as timeslots available, as in Brazil [16,17], Italy [8,19], Spain [3], Switzerland [10], and the UK [28]. In cases with optional subjects it is usual impossible to have compact schedules. These schedules, on the student level now, are not often considered for quality.

2. Compact schedules for teachers. This is mentioned in most of the papers above. Sometimes, as with classes, this is almost automatic (this assumes that all, or at least almost all, teachers work full-time).

Considering all cases above, it is difficult to judge which problems are harder than others. Probably most cases have aspects that are difficult to handle; either feasibility is hard to attain, or it can be hard to obtain high quality schedules. Similarly it is difficult to judge the capabilities of different methods; how well would a method do on cases outside the studied ones? The lack of exchangeable benchmarks is obvious here.

As for the Netherlands, we think that the high school timetabling problem is quite difficult; we will try to explain that in the next section. Since De Gans [12] in 1981 no study seems to be published internationally on real-life data from the Netherlands. (Willemens [26] focuses on complexity issues.) Clearly a lot has changed since 1981, and it is worth studying the situation as it stands now. The process is still very dynamic, see Section 6.

## 2 Problem Description

The impetus for this research was a request in 2003 from the Kottenpark (a location of ‘Het Stedelijk Lyceum’ in Enschede) to assist with creating the year schedule. In the Kottenpark the timetable is still mainly made by hand, and checked by computer. The reason for not using the commercial engine is mainly quality: the engine is not able to generate any complete solution, and the part that is generated is of poor quality.

In 2004, the Kottenpark had around 1000 students, 36 school classes, 71 teachers, and 40 rooms. There were 1049 lessons to be scheduled. As such it is a school of average size in the Netherlands. There are 38 time slots available for lessons. The occupation of time slots by the students ranges from 76% (29 lessons per week) up to 92% (35 lessons per week). In timetabling this school the following difficulties are encountered.

- In the upper years, up to two-thirds of the lessons of the students are in optional subjects. To handle these subjects, a cluster scheme (see below) is constructed, which requires a certain number of time slots (the length of the cluster scheme). It is sometimes difficult to reduce this length to get an a priori schedulable situation.
- Around 75% of the teachers work part-time at the Kottenpark. Consequently (by collective labor agreement) they are entitled to have 1, 2 or 3 days without lessons; usually teachers have preferences for these days.
- Teachers have up to 26 lessons, most of them less. Hence avoiding idle time is not automatic. The hand-made schedule contains 128 idle times for teachers, an average of 1.9 per teacher.
- The lower years (without optional subjects) should have no idle times.
- The two gymnasiums are used for 100% of the time, often as a block (two consecutive time slots), and always two classes (of the same age) combined.

All these circumstances are quite common in the Netherlands. It seems that scheduling the students, and the high amount of part-time teachers are quite exceptional compared to other countries. Our approach consists of three phases, devised to handle these problems:

1. *Construct cluster schemes*: the students are assigned to classes for the optional subjects, and the classes to cluster lines.
2. *Create a feasible schedule*: assign all lessons to time slots, such that there are no clashes.
3. *Improve the schedule*: improve the feasible schedule.

In the next sections, we will describe the three phases of our algorithm in more detail.

### 3 Constructing the Cluster Schemes

#### 3.1 Motivation

As one can easily imagine, scheduling the optional subjects constitutes a major bottleneck in constructing (good) timetables. To get this process under control, *cluster schemes* are constructed first. In the Netherlands it seems that all schools first construct cluster schemes. This is a way to avoid using individual students, essential when the schedule is constructed by hand. The system with optional subjects was introduced in 1968 (Mammoetwet), while the revision of 1995 ('Second Phase') made things even more complicated. The report of Simons [20] concerns the construction of cluster schemes, while De Gans [12] assumes that the cluster schemes are already created.

We have to construct a cluster scheme for years with optional subjects. Each cluster scheme consists of *cluster lines*. Each cluster line contains a number of classes with different optional subjects, that will be scheduled at the same time slots. The arrangement of classes in the cluster lines, and the assignment of students to classes of their optional subjects must be such that each student can attend the optional subjects he has chosen. So for each student it should be possible to make an assignment to classes of his optional subjects, such that these classes are in different cluster lines. Unfortunately, not all optional subjects have the same number of lessons, neither have all students the same number of optional subjects. Homogeneous tiling structures, as in [14], seem difficult to attain. (The high percentage of part-timers also breaks homogeneity.)

The classes in a cluster line have a number of lessons. The maximal number of lessons in a cluster line is called the *cluster line length*. The *cluster scheme length* is the sum of the lengths of the cluster lines it contains; it represents the number of time slots we have to reserve for the cluster scheme.

The main goal is to minimize the cluster scheme lengths. (Commercial software prescribes the maximal number of cluster lines, which is less informative.) For this there are the following reasons.



- If the cluster scheme length is too high, it might be that there are not enough time slots left for the obligatory subjects. In some years this causes a problem. It can even happen that the school management decides to add an extra class to decrease the cluster scheme length.
- A lower cluster scheme length increases the freedom in scheduling the remaining obligatory subjects.
- A lower cluster scheme length decreases the number of potential idle times. Here it is important to remember that not all students might be present in a cluster line. For the time slots of this cluster line these students are free. If these students have a lesson earlier and later on the day, it is an idle time.

The secondary goal in a cluster scheme is to balance the classes of a subject. If, for instance, there are three classes for the subject mathematics, and 79 students, then the best balance is that the classes contain 26, 26 and 27 students. The combination 23, 23 and 33 is forbidden, as this violates the maximum size (32 in our case) of a class, while the combination 32, 32, 15 is not desirable. Balancing is done for educational reasons, and for fairness towards the teachers.

### 3.2 Branch and Bound

The sizes of problem instances (up to 20 or 25 classes) make it worthwhile to attempt careful enumeration. If we have 20 classes and 8 cluster lines, then a priori there are  $8^{20}$  possibilities for classes in lines, which is clearly out of range. However there are several ways to reduce the number of possibilities considerably. In the enumeration there are two possible approaches.

1. Decide per step in which classes a student takes his optional subjects. In this case we have to put the classes in lines, such that classes in a line have no students in common. This can be viewed as a graph coloring problem (classes are vertices, with edges between non-disjoint classes, and cluster lines are colors).
2. Decide per step in which lines the classes are put. In each step we have to solve a matching problem for each student, namely to decide in which lines a student takes his optional subjects.

Both approaches will try to extend partial assignments, and step to the next possibility in case the configuration gives no solution. We chose the second approach for two reasons:

- It seems easier to solve at each step a matching problem than a graph coloring one.
- It seems easier to estimate our objectives, the cluster scheme length and the balance of classes.

Hence our approach takes the following steps. We store, for instance, the best 100 solutions:

1. Place one student in a group for his  $k$  subjects, and place these groups in the first  $k$  cluster lines. These groups are fixed, and will never be moved.

2. Take the first not yet placed group  $G$ .
3. For cluster line 1 to the last cluster line, place  $G$ , and estimate the cluster scheme length. If too high skip the case, and move  $G$  to the next cluster line.
4. Assign all students to the placed groups (increment for  $G$ ). If not all students can be assigned, stop the case and move  $G$  to the next line.
5. If there are non-placed groups left, return to 2.
6. Balance the groups, and calculate the objectives, the cluster scheme length and the penalty for not-balanced groups. We compare solutions lexicographically, first by cluster scheme length, and then by balance. If necessary store the solution. Continue with placing  $G$  in the next cluster line.

We give an overview of the methods we applied to speed up the search process. For more details we refer to the technical report [15].

### 3.3 Using Statistics

Calculating the matchings at each step is very time consuming. A first simplification is to group students that have the same optional subjects. Apart from this, we accumulate some statistics at the start. We will call an optional subject with  $k$  classes a ' $k$ -grouper'.

- For any two 1-groupers, check if they have a student in common. If this is the case, the corresponding classes have to be placed in different lines.
- For any two 1-groupers, and a 2-grouper, check if there is a student with this combination of subjects. If this is the case, the corresponding 4 classes have to occupy at least 3 lines.

To use these statistics in an efficient way, we decided to order the classes according to the number of classes of the subject; the 1-groupers first, then the 2-groupers, then the 3-groupers, and so on. Note that placing the 1-groupers is a graph coloring problem: the classes are the vertices of the graph, while two vertices are connected if the corresponding classes have a student in common (this is part of the statistics above).

When placing a 2-grouper, we similarly use the statistics. At the moment we try to place the second class of this subject, we collect all 1-groupers in the two corresponding lines. If two 1-groupers with the 2-grouper is chosen by a student (this is in the statistics), the combination is forbidden, and does not need consideration.

### 3.4 Symmetry

All lines are equivalent. Hence in case of eight lines, we gain a factor 8! by removing equivalent solutions. We remove symmetric solutions by:

- Fixing one (difficult) student to classes, and fixing these classes in different lines.
- Only place a class in a cluster line, when all previous lines are non-empty.

- For two classes of the same subject we assume that the line number of the first class is lower than the line number of the second class. This holds if these classes that are not fixed by the difficult student (as described above).

Even this does not remove all symmetry. The situation that can occur is that a subject  $S3$  with (say) two classes is fixed in line 1. The subjects  $S1$  and  $S2$  are placed before the non-fixed class of  $S3$  is. Then the following can happen:

$$\begin{array}{c} \text{line 1: } S3 \ S1 \\ \hline \text{line 2: } S2 \ S3 \end{array} \quad \text{and} \quad \begin{array}{c} \text{line 1: } S3 \ S2 \\ \hline \text{line 2: } S1 \ S3 \end{array}$$

If the fixed class of  $S3$  is not there, the second solution is forbidden; line 1 would still be empty at the moment we start to place subject  $S1$ . It seems hard to avoid this kind of symmetry.

### 3.5 Bounding

The next part we have to take care of is bounding. In our problem we have two things to bound on: first the length of the cluster scheme, and second the balancing of the classes.

The length of the cluster scheme can be estimated by the classes that were placed in lines. In the case that not all classes are placed yet, we have a lower bound for the cluster scheme length, which can be used for bounding: as soon as the partial cluster scheme has a length exceeding the best obtained cluster scheme, we prune the search tree.

At the moment we start to place the classes of a new subject  $j$ , we place it in line  $i$ , and calculate (by matching) the maximum number of students  $M_{ij}$  that can be assigned. In the partial cluster scheme this is an exact calculation; this number however, can decrease when new subjects are placed. As soon as all classes of a subject are placed in cluster lines, we can estimate how far the classes necessarily will deviate from the average size; again we prune if the deviation is higher than the best found.

### 3.6 Balancing Heuristic

Once a *complete* cluster scheme has been found, and all classes have been put in cluster lines, the assignments of students to classes have to be reconsidered. The assignment were made by the Hungarian method, and hence by first fit; no attention was paid to the number of students in the classes. In particular we prefer that classes of the same subject contain approximately the same number of students. Our heuristic for balancing is a greedy algorithm; here  $M_{ij}$  is as above, and  $A_j$  denotes the average size for classes of subject  $j$ .

1. Find the line  $i$  and subject  $j$  where  $M_{ij} - A_j$  is negative and minimal. We assign as many as possible students to this class, and discard the combination  $(i, j)$  in the sequel. We continue until all  $(i, j)$  with  $M_{ij} < A_j$  are treated.

2. If for remaining combinations  $(i, j)$  we have that  $M_{ij} - A_j$  is non-negative, we turn our attention to classes which are still below average, and proceed in the same way, with  $M_{ij}$  replaced by the number of currently assigned students.

We could continue with balancing of the classes above average, but we do not do so. In practice there seems to be no need for it.

### 3.7 Pruning Based on Computation Time

The program we developed contains an option to prune parts of the search tree, based on the time spent in the subtree: one can prescribe that for search depth  $d$  only  $s$  seconds are allowed. Here  $d$  is usually between 1 and 4, while  $s$  is taken as a few seconds. In this way we can do a quick scan of the search space within one or two minutes. Especially for the harder cases, it turns out that solutions are found much quicker this way.

### 3.8 Results

The methods above have been used at the Kottenpark during the last three years. When running the program an upper bound for the cluster scheme length must be given. For most years good solutions are found within a few seconds, if at least one feasible solution exists. If no solutions exist, the search can take several minutes or hours, as the complete search tree has to be checked.

## 4 Creating a Feasible Schedule

### 4.1 Motivation

Schools consider the creation of the cluster schemes as a preliminary phase; usually a different application is provided for it, without interaction with the timetabling itself. Once the cluster schemes are found, the next phase starts to assign the lessons to time slots. Instead of assigning the lessons directly to time slots, we will first assign them to day-parts. For this the days at Kottenpark are divided in two day-parts: mornings of five time slots, and afternoons with three time slots, except for the Thursday afternoon, which consists of one time slot. There are several reasons to do this intermediate step.

- Several constraints are on daily, or on day-part level. These are the constraints that lessons of a class must be on different days, teachers are not available on certain days or day-parts, and the number of teaching days for teachers should be limited for part-time teachers. Such constraints can be handled very well in this phase.
- It is unclear to which time slot we should assign a certain lesson, if we do not know about all lessons to be assigned. Hence we will do a lot of useless reshuffling.

- Assigning lessons to 40 time slots in a week is much harder than assigning lessons to the 5 or 3 time slots in a day-part for 10 times.

Of course there are certain drawbacks. The most important one is that a feasible day-part schedule does not imply a feasible time slot schedule. This problem we address in Section 4.3. Moreover we restrict the search space; while mentioned above as an advantage, it could prevent us from improving certain aspects in the solution.

## 4.2 Direct Heuristic

We proceed to schedule the lessons to day-parts. To do this, the lessons are grouped by the class, or for the optional subjects, by the cluster line. For uniformity we create an artificial cluster line (with one class) for the compulsory subjects. Hence in a stage of the direct heuristic we consider a cluster line, and try to assign the lessons of the cluster lines to day-parts.

The method we use is a dynamic priority rule. At each stage we estimate the difficulty of the cluster lines to be scheduled. The difficulty is based on the weight of the cluster line (originally all weights are 0), and the availabilities of the resources of the cluster line. We will schedule the most difficult cluster line first. If this scheduling process breaks down, because a particular cluster line cannot be assigned any more, we raise the weight of this cluster line, and restart.

## 4.3 Compatibility Checking

If we assign cluster lines to day-parts, some conditions have to be checked. The obvious necessary conditions are that a resource is scheduled for at most the number of time slots that it is available. This, however, is not enough to guarantee schedulability on time slot level. We can take certain measures which in practice are sufficient. These measures consist of creating time slot schedules for day-parts and resources that get tight. More specifically, if for a resource the slack in time slots is 1 or 0, we decide to do this check. In that case, all lessons of this resource are taken, as well as the neighbors, and the neighbors' neighbor (the compatibility graph). Here a neighbor is defined to be a lesson with a common teacher, class, or student. We try to color this graph where the colors are the available time slots. If we do not succeed within a certain time limit, we assume that no coloring exists, and reject the day-part for this lesson.

Here some special attention has to be taken towards the resource 'room type'. (We do not really schedule the rooms, but only make sure that we have enough rooms of the required room type available.) The lessons with the required room types are not necessarily neighbors in the compatibility graph. Hence, when constructing the subgraph, we take all lessons with the required room type, and add all neighbors and neighbors' neighbors.

## 4.4 Assigning the Time Slots

Once all lessons have been assigned to day-parts, we try to assign them to time slots. For this we use a graph coloring heuristic, which colors the nodes one by one (first fit). To sort the nodes, we use the weight of the nodes, where the weight is originally the degree of the node. Each time a node cannot be colored any more, we increase the weight of this node, and backtrack.

## 4.5 Results

Thanks to the checks described in Section 4.3 all lessons are scheduled after a few restarts (see Section 4.2), which takes a few minutes. The quite extensive compatibility tests turn out to be beneficial on the running time. We tried to influence the coloring with regard to idle times for teachers. For this we also started off with random orderings of the nodes (see Section 4.4), instead of ordering by degree. Allowing 30 seconds per day-part for this random search reduced the total number of idle times for teachers from 142 to 114. Repeating the random searches 20 times, the total number of idle times dropped to 95. Hence on this aspect we beat the hand-made schedule. Unfortunately there are still idle times for the classes of the lower years, which we did not take into account.

# 5 Improving the Schedule

## 5.1 Motivation

The previous phase aimed at assigning all lessons to time slots. Not much attention was paid to quality yet; the emphasis was on finding a feasible schedule. In the current phase, we try to improve the feasible schedule we found. Ejection chains [2] combined with tabu search [13] seem to be very appropriate for improving schedules. The quality of a schedule is determined per resource by the idle times, and the division of lessons over the days for the resource. Hence we can find a resource, with a low quality schedule, improve this schedule by shifting some lessons, and prevent shifting back by placing this shift on a tabu list. An improvement for one resource (teacher/class) automatically implies that for other resources (class/teacher) some repairs have to be done; the shifted lessons can have clashes, due to other resources. Such lessons, ‘conflict-lessons’, again have their own conflicts, etc. Hence we run quite naturally into an ejection chain of improvements.

## 5.2 Selecting the Shifts

We perform a tabu search, in which each step by itself is a chain of shifts. Here a shift means that a lesson is moved from one time slot to another. We explain how we find the shifts that we execute.

1. First we select the resource A (teacher or class) with the worst schedule, which is not tabu.
2. For resource A we consider all lessons and all free time slots. For each combination (lesson, free time slot) we calculate the cost change for resource A for moving a lesson to a free time slot. With respect to other resources of the lesson, we only make sure that there is not more than one conflict lesson.
3. We select the  $C = 20$  best candidates. The selected shifts we call the *first shift candidates*.
4. Executing the first shift candidates, two things can happen:
  - The other resources have no clashes; in this case the chain of shifts is ended.
  - There is one conflict-lesson (we did not allow more than one!), due to resource B. We shift the conflict-lesson to a free time slot of resource B, and calculate what is the best for resource B. Again we only consider time slots with at most one conflict-lesson; time slots without conflict-lesson are preferred.

If in the second case no new conflict arises, then the chain of shifts is ended. Otherwise we proceed until a maximum of  $D = 10$  shifts.

5. The chain with the highest cost reduction is executed; the reverse move of first shift is made tabu for  $L = 10$  moves. If no chain is found at all, resource A itself is made tabu also for  $L$  moves.

### 5.3 Results

We let the algorithm run for 2500 iterations. Usually the best schedule is found within 1500 iterations. With the parameters as above the algorithm runs for less than one minute. Experiments were executed with different sets of constraints. In case of the default Kottenpark set, the cost is reduced by more than 70%. The total number of idle times of the teachers reduced to 48, while the idle times of the lower years disappeared. Moreover the spreading of lessons for the teachers was improved considerably.

## 6 Conclusion

The presented study is performed with data from a specific Dutch school, but we believe that these data are representative for many schools in the Netherlands. Unfortunately not all constraints are incorporated yet, which makes comparison to the real timetable not completely fair. Comparing what *is* included we see a huge improvement in quality; for instance the number of free periods for teachers drops from 128 (hand-made) to 48, maintaining compact schedules for the lower years.

After constructing the cluster schemes, we used a two-phase approach to obtain feasible schedules; the first phase (Section 4) was designed to handle several constraints related to part-time teachers. Viewing the result in the second phase

(Section 5), one can wonder at the effectiveness of this approach; many lessons were moved from one day to another, improving the spreading of lessons for part-time teachers. Nevertheless the two-phase method is quite effective in obtaining a feasible schedule.

In 2005, the Kottenpark introduced a new educational system in the two lower years. In this system classes of 60 students are constructed. Most subjects are taught with two teachers: the first teacher belongs to the subject (preassigned as before), while the second teacher is one of the two preassigned to the class. The 24 lessons of one class have to be divided between these two teachers, where there is some preference for subjects, but split assignments are allowed. Because of this, the program as described here is used operationally only for constructing the cluster schemes.

*Acknowledgements.* This research has been supported by the Netherlands Organization for Scientific Research, grant 636.000.000.02N18 (Leraar in Onderzoek), and by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

## References

1. Abramson, D.: Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science* 37, 98–113 (1991)
2. Ahuja, R.K., Ergu, O., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. *Discrete and Applied Mathematics* 123, 75–102 (2002)
3. Alvarez-Valdes, R., Martin, G., Tamarit, J.M.: Constructing good solutions for the Spanish school timetabling problem. *Journal of Operational Research Society* 47, 1203–1215 (1996)
4. Bufé, M., Fischer, T., Gubbels, H., Häcker, C., Hasprich, O., Scheibel, C., Karsten Weicker, K., Weicker, N., Wenig, M., Wolfangel, C.: Automated solution of a highly constrained school timetabling problem – preliminary results. In: Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tjijink, H. (eds.) *EvoIASP 2001, EvoWorkshops 2001, EvoFlight 2001, EvoSTIM 2001, EvoCOP 2001, and EvoLearn 2001*. LNCS, vol. 2037, pp. 431–440. Springer, Heidelberg (2001)
5. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002)
6. Birbis, T., Daskalali, S., Housos, E.: Timetabling for Greek high schools. *Journal of the Operational Research Society* 48, 1191–1200 (1997)
7. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
8. Colorni, A., Dorigo, M., Maniezzo, V.: Metaheuristics for high school timetabling. *Computational Optimization and Applications* 9, 275–298 (1998)
9. Cooper, T.B., Kingston, J.: The solution of real instances of the timetabling problem. *The Computer Journal* 36, 645–653 (1993)
10. Costa, D.: A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research* 76, 98–110 (1994)



11. Drexl, A., Salewski, F.: Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research* 102, 193–214 (1997)
12. de Gans, O.B.: A computer timetabling system for secondary schools in the Netherlands. *European Journal of Operational Research* 7, 175–182 (1981)
13. Glover, F.W., Laguna, M.: *Tabu Search*. Kluwer, Norwell, MA (1997)
14. Kingston, J.H.: A tiling algorithm for high school timetabling. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 208–225. Springer, Heidelberg (2005)
15. Post, G.F.H., Ruizenaar, W.A.: Clusterschemes in Dutch secondary schools. Memorandum 1707, University of Twente (2004), <http://www.math.utwente.nl/publications/2004/1707abs.html>
16. Ribeiro Filho, G., Lorena, L.A.N.: A constructive approach to school timetabling. In: Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tjink, H. (eds.) *EvoIASP 2001, EvoWorkshops 2001, EvoFlight 2001, EvoSTIM 2001, EvoCOP 2001, and EvoLearn 2001*. LNCS, vol. 2037, pp. 130–139. Springer, Heidelberg (2001)
17. Santos, H.G., Ochi, L.S., Souza, M.J.F.: An efficient tabu search heuristic for the school timetabling problem. In: Ribeiro, C.C., Martins, S.L. (eds.) *WEA 2004*. LNCS, vol. 3059, pp. 468–481. Springer, Heidelberg (2004)
18. Schaerf, A.: A survey of automated timetabling. CWI Report CS-R9567, CWI, The Netherlands (1995)
19. Schaerf, A.: Local search techniques for large high school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans* 29, 368–377 (1999)
20. Simons, J.L.: ABC: Een programma dat automatisch blokken construeert bij de vakdifferentiatie binnen het algemeen voortgezet onderwijs. Technical Report NLR TR 74107 U, NLR, The Netherlands (1974)
21. Smith, K.A., Abramson, D., Duke, D.: Hopfield neural networks for timetabling: formulations, methods and comparative results. *Computers and Industrial Engineering* 44, 283–305 (2003)
22. Valouxis, C., Housos, E.: Constraint programming approach for school timetabling. *Computers and Operations Research* 30, 1555–1572 (2003)
23. de Werra, D.: An introduction to timetabling. *European Journal of Operational Research* 19, 151–162 (1985)
24. de Werra, D.: On a multiconstrained model for chromatic scheduling. *Discrete Applied Mathematics* 94, 171–180 (1999)
25. Wilke, P., Gröbner, M., Oster, N.: A hybrid algorithm for school timetabling. In: McKay, B., Slaney, J.K. (eds.) *AI 2002: Advances in Artificial Intelligence*. LNCS (LNAI), vol. 2557, pp. 455–464. Springer, Heidelberg (2002)
26. Willemen, R.J.: School timetable construction; algorithms and complexity, Ph.D. Thesis, Technical University Eindhoven, The Netherlands (2002)
27. Wood, J., Whitaker, D.: Student centred school timetabling. *Journal of Operational Research Society* 49, 1146–1152 (1998)
28. Wright, M.: School timetabling using heuristic search. *Journal of Operational Research Society* 47, 347–357 (1996)

# Scheduling School Meetings

Franca Rinaldi and Paolo Serafini

Department of Mathematics and Computer Science, University of Udine,  
Via delle Scienze 206, 33100 Udine, Italy  
{rinaldi,serafini}@dimi.uniud.it

**Abstract.** Prespecified meetings between teachers and parents have to be scheduled. All meetings have the same duration. The goal is in finding a schedule minimizing the total time and the parents' idle times. This NP-hard problem is addressed by solving first a sequence of weighted assignment problems and then performing a large scale neighborhood search based on finding negative cost cycles and shortest paths in directed graphs. This approach provides good computational results. Finally a variant of the problem with two different durations for the meetings is considered.

## 1 Introduction

We address the following problem arising in Italian high schools: on certain days of the school year, parents can meet teachers to discuss their children. Each parent tries to meet some teachers and the meetings are individual. There is no advance planning for the event and therefore parents wait in lines for a long time (one line for each teacher), not only wasting time but also preventing the possibility of meeting several teachers.

In this paper we propose a planning method in order to guarantee that each parent meets all required teachers and the global wasted time of the parents is minimized. A prerequisite for the method to work is that all meetings last the same amount of time.

The problem we study is NP-hard. We suggest a two-phase heuristic approach for its solution. The first phase computes a schedule of minimum time to allocate all meetings by solving a sequence of weighted assignment problems. The second phase minimizes the parents' idle times through a large-scale neighborhood search based on negative cycle detection and shortest path computations in directed graphs. It turns out that this search is quite effective and strongly reduces the wasted time of the final solutions.

We also briefly consider an extension of the model by allowing double duration to certain specified meetings that require more discussion.

To the best of our knowledge, this specific timetable problem has not been addressed in the literature. Nevertheless, a similar problem was considered in [3] to schedule job interviews for law firms and students at the Southeastern Public Interest Job Fair, a law fair which is held each year in the USA. In [3] the timetable problem is modeled as an edge-coloring problem on bipartite graphs

and the minimization of the idle times (considered both for law firms and students) is partially carried out by maximizing the cumulative number of meetings assigned to each period. Our problem has also some resemblance with the no-wait open shop problem [8], where it is required that all the operations of a same job are executed contiguously, i.e. no idle time for the jobs is allowed. The particular case of unit processing times has been studied in [6] and, under the additional condition of no idle time for the machines, in [5].

As pointed out above, the local search procedure we adopt is based on neighborhoods of exponential size in which an improving neighbor can be found in polynomial time by dynamic programming. The potential of using optimization methods, and in particular network flow techniques and dynamic programming, to search neighborhoods of very large size, has been remarked on in [1]. In the particular field of timetabling, large-scale neighborhood metaheuristics have been applied, for instance, in [4] to solve a nurse scheduling problem and in [9] to solve the examination timetabling problem.

The paper is organized as follows. In Section 2 we define the problem. In Section 3 an algorithm to find a schedule of minimum makespan is presented. In Sections 4 and 5 two local search procedures based on finding sequences of meeting exchanges are described. The first one considers meetings of a fixed teacher, whereas the second one considers meetings of a fixed parent. The techniques are illustrated with an example in Section 6. In Section 7 we extend the problem to the case with two different durations of meetings. Finally some computational results are given in Section 8.

## 2 Problem Definition

In the problem we consider, a set  $J$  of parents and a set  $I$  of teachers are given and each parent  $j$  wants to meet a specified subset  $I_j$  of teachers. The subsets  $I_j$  are the input data. Let  $J_i := \{j : i \in I_j\}$  be the subset of parents that want to meet the teacher  $i$ .

Assuming that all meetings last the same amount of time, called *time slot*, the output is a schedule  $t(i, j)$  assigning the time slot  $t(i, j)$  to the meeting of parent  $j$  with teacher  $i$ , with the obvious requirement that  $t(i, j) \neq t(i, j')$  for all  $j' \neq j$  and all  $i$  and also  $t(i, j) \neq t(i', j)$  for all  $i' \neq i$  and all  $j$ . We define as *makespan* of the schedule the maximum time in which a meeting occurs: i.e.,  $\max_{i \in I} t(i, j) = \max_{j \in J} t(i, j)$ . Moreover, for any given schedule, we define *idle time* for parent  $j$  any time slot  $k$  such that  $\min_i t(i, j) < k < \max_i t(i, j)$  and  $k \neq t(i, j)$  for any  $i$ . In other words, an idle time is a waiting time slot in between meetings and therefore counts as a wasted time.

We define as *school meeting problem* the problem of determining a schedule of minimum makespan that minimizes the total number of idle times over all parents. In this way the two objectives of minimizing the makespan and the wasted time of the parents are treated in a lexicographic order. The problem of minimizing the number of idle times within a fixed time is NP-hard, as can be seen by transforming the no-wait open shop problem with 0–1 processing

times  $O|\text{no-wait}, p_{ij} \in \{0, 1\} | C_{\max}$ , shown to be NP-hard in [6]. Then the school meeting problem is NP-hard. We approach its solution heuristically.

### 3 Minimizing the Maximum Time for the Meetings

The problem of minimizing the maximum time needed for the meetings is equivalent to a minimum makespan open-shop problem by identifying teachers with machines and parents with jobs. Since the processing times are equal to one, this particular instance of the open shop problem is polynomial and can be easily solved by means of the algorithm by Gonzalez and Sahni [7] for the open-shop problem with pre-emption.

The minimum makespan  $T$  is given by  $T = \max \{ \max_j |I_j|, \max_i |J_i| \}$ , and an optimal schedule can be computed by solving a sequence of  $T$  bipartite matching problems, one for each time slot. In more detail, let  $J_i^0 := J_i$  and  $I_j^0 := I_j$ . Then recursively the  $k$ th matching problem ( $k = 1, \dots, T$ ) assigns a teacher  $i$  to a parent  $j$  only if  $i \in I_j^{k-1}$ . Furthermore, teachers  $i$  such that  $|J_i^{k-1}| = T - k + 1$  must be assigned to some parent and parents  $j$  such that  $|I_j^{k-1}| = T - k + 1$  must be assigned to some teacher. These teachers and parents are called *critical* for time  $k$  (and remain critical for any subsequent time). Let  $M^k \subset I \times J$  be the set of pairs assigned by the  $k$ th matching. Then for all  $(i, j) \in M^k$ , set  $t(i, j) := k$ ,  $I_j^k := I_j^{k-1} \setminus \{i\}$  and  $J_i^k := J_i^{k-1} \setminus \{j\}$ .

The structure of the above algorithm leaves some room to take into account the lexicographically second objective function of the school meeting problem, that is minimizing the total number of idle times of the parents. Our first strategy consists in solving, at each step, a max weight matching problem instead of a feasible matching problem. To this aim let  $s_k(j) \in \{0, 1, 2\}$  be the state of parent  $j$  at time  $k$ , where the meaning is that  $s_k(j) = 0$  if no meeting has been assigned to parent  $j$  up to time slot  $(k - 1)$  (included),  $s_k(j) = 1$  if some, but not all, meetings have been assigned, and  $s_k(j) = 2$  if all meetings have been assigned. Then each pair  $(i, j)$  receives the weight

$$w_{ij} := \begin{cases} 0 & \text{if } s_k(j) = 0 \\ 1 & \text{if } s_k(j) = 1 \end{cases} \tag{1}$$

in the  $k$ th matching problem (note that if  $s_k(j) = 2$  parent  $j$  is not considered in the matching problem). Hence, until a parent has been assigned, his/her meetings receive a zero weight. As soon as a meeting for the parent is assigned, the weight rises to one.

With the cost function (1) the procedure tends to schedule the major part of the meetings in the last slots, when all parents and teachers become critical. This introduces inevitable idle times. In order to overcome this behavior, we have observed that it is useful to modify the cost function by randomly generating the meeting costs so that parents still to be assigned might be encouraged to be assigned a meeting. So we redefine (1) as

$$w_{ij} := \left. \begin{cases} -1 & \text{with probability } p_1 \\ 0 & \text{with probability } p_2 \\ 1 & \text{with probability } 1 - p_1 - p_2 \end{cases} \right\} \text{ if } s_k(j) = 0 \tag{2}$$

$$K(1 + W_k(j))^2 \quad \text{if } s_k(j) = 1$$

with  $W_k(j)$  the number of idle times assigned to parent  $j$  during time slots  $\{1, \dots, k - 1\}$ . The probabilities  $p_1$  and  $p_2$  and the coefficient  $K$  must be properly tuned. Hence, until a parent has been assigned, his/her meetings receive a low weight, not greater than one. As soon as a meeting for the parent is assigned, the weight rises to  $K$  and then increases quadratically with the number of assigned idle times.

### 4 Minimizing the Idle Times – Local Search LST

The solution found by the procedure described in the previous section may have many idle times. In order to reduce their number, we adopt a large-scale neighborhood search approach based on two different types of neighborhood. The first procedure, denoted LST, modifies the schedule by exchanging meetings of a single teacher, while the second procedure, denoted LSP, moves meetings of a single parent. In this section we present the local search LST.

This local search is based on the idea of moving a given meeting to a time slot in which the parent is free. If in the new time slot the teacher is busy with another meeting, this meeting must be moved as well, creating recursively a chain of moves. For feasibility, this chain of moves must either be cyclic or end in a free time slot for the teacher. We associate to this chain a cost given by the number of introduced idle times. Although the number of possible moves is exponential, an improving solution in the neighborhood, if any exists, can be found in polynomial time as follows.

Given a schedule with makespan  $T$  and a particular teacher  $i$ , we define a directed graph  $G_i = (N, E_i)$  having  $T$  nodes identified with the time slots  $1, \dots, T$ .

Let  $B_i$  be the set of nodes corresponding to the time slots when teacher  $i$  is busy with some meeting, i.e.  $B_i := \cup_{j \in J_i} t(i, j)$  and  $F_i := N \setminus B_i$  be the complement set of nodes when teacher  $i$  is free from meetings. Moreover, for each parent  $j$ , let  $H_j$  be the set of nodes corresponding to the time slots when  $j$  is not assigned a meeting, i.e.  $H_j := N \setminus \cup_{i \in I_j} t(i, j)$ . Note that  $H_j$  does not depend on  $i$ .

The arcs  $E_i$  of the graph  $G_i$  are defined as follows. For each  $k \in B_i$ , let  $j(k)$  be the parent meeting  $i$  at time  $k$ , i.e.  $t(i, j(k)) = k$ . There is an arc  $(k, h)$  for each  $h \in H_{j(k)}$ . This arc corresponds to moving the meeting of teacher  $i$  and parent  $j(k)$  from time  $k = t(i, j(k))$  to time  $h$ . There is no conflict for the parent because the parent is free at time  $h$ . There is a conflict for the teacher if  $h \in B_i$ . But this conflict can be resolved by a subsequent change induced by another arc. Therefore any simple cycle in  $G_i$  corresponds to a sequence of changes which eventually lead to a new feasible schedule. Similarly, any simple path terminating in a node  $h \in F_i$  corresponds to a feasible sequence of changes. We may extend paths to cycles by adding arcs  $(h, k)$  for each  $h \in F_i$  and  $k \in B_i$ .

We assign the following costs to the arcs: the arcs  $(h, k)$ ,  $h \in F_i, k \in B_i$  receive cost 0; the arcs  $(k, h)$ ,  $k \in B_i, h \in H_{j(k)}$  receive a cost given by the difference between the idle time number for parent  $j(k)$  in the current schedule and in the schedule obtained by moving the meeting from  $k = t(i, j(k))$  to time  $h$ . Then simple cycles with negative cost correspond to a sequence of changes leading to a schedule with fewer idle times. Detecting negative cost simple cycles is easy and can be carried out for instance by the Floyd–Warshall algorithm (see [2]). We point out the similarity of this type of search with the one proposed in [4].

The above considerations suggest the following scheme for a large-scale neighborhood search: given a schedule, for each teacher  $i$  build the graph  $G_i$  and detect a negative simple cycle  $C_i$ . If there is no such cycle output a shortest directed cycle. Then select the  $i$  with the most negative cycles and among them randomly select a particular  $i'$ . If there are no negative cycles, select randomly any  $i'$  with zero cycle cost if any exists. If there are only positive cost cycles stop the procedure, else perform the meeting exchanges indicated by the cycle and continue. Stop the procedure if the number of consecutive zero cost exchanges exceeds a fixed parameter.

### 5 Minimizing the Idle Times – Local Search LSP

In the local search LSP we exchange the roles of teachers and parents. Therefore we move a given meeting to a time slot in which the teacher is free. If in the new time slot the parent is busy with another meeting, this meeting must be moved as well. However, since we count idle times for parents and not for teachers, there is not full symmetry between the two approaches. In order to count the number of idle times introduced or removed by a chain of moves, note that the number of idle times of a single parent changes if and only if his/her total time in the school changes.

The procedure works as follows. Let  $G_j$  a graph having nodes corresponding to time slots  $1, \dots, T$  and arcs  $(k, h)$  whenever  $j$  meets a teacher, let us say  $i$ , at time  $k$  and  $i$  is free at time  $h$ . Let  $t' := \min_i t(i, j)$  and  $t'' := \max_i t(i, j)$ . By the above comment, a sequence of switching in the meetings of  $j$  changes the number of idle times for  $j$  if and only if  $t'' - t'$  changes.

In order to find a sequence that reduces the number of idle times of parent  $j$ , we solve two shortest path problems in  $G_j$ , one from  $t'$  and one from  $t''$  to the set  $H_j$  of nodes at which parent  $j$  is free. The instance with source  $t'$  is defined as follows. Let  $r \geq 0$  be the number of idle time slots adjacent to  $t'$  and  $B_j = \cup_i t(i, j)$ . We assign to the arcs of  $G_j$  the following costs:

$$w(k, h) := \begin{cases} -(h - t') & \text{if } k = t', h \leq t' + r \\ -(r + 1) & \text{if } k = t', h \in H_j, t' + r < h < t'' \\ -r + s - 1 & \text{if } k = t', h = t'' + s > t'' \\ -r & \text{if } k = t', h \in B_j \\ r - (h - t') & \text{if } k \in B_j \setminus \{t'\}, h \leq t' + r \\ s - 1 & \text{if } k \in B_j \setminus \{t'\}, h = t'' + s > t'' \\ -1 & \text{if } k \in B_j \setminus \{t'\}, h \in H_j, t' + r < h < t'' \\ 0 & \text{if } k, h \in B_j \setminus \{t'\}. \end{cases} \tag{3}$$

Then the cost of a directed path from  $t'$  to any node in  $H_j$  measures the change in the number of idle times produced by the corresponding sequence of moves. This is clearly the case if the path has only one arc  $(t', h)$  (first three cases in (3)). If the path has more than one arc, then all arcs different from the first and the last ones correspond to exchanges in between meetings on the same time slots and do not modify the number of idle times. Consistently, they have a null cost (last case) and the change in the idle time number depends only on the first and last arc. The first arc always eliminates  $r$  idle times (fourth case). The last arc  $(k, h)$  eliminates one more idle time if  $h$  is an idle time, while it may introduce idle times if either  $h \leq t' + r$  or  $h \geq t'' + 1$  (remaining cases). Note that there are no negative cycles and so a shortest path problem is well defined. The instance of the shortest path problem with source  $t''$  can be defined in a similar way.

We define a large-scale neighborhood search scheme for LSP as for LST: given a schedule, for each parent  $j$  build the graph  $G_j$  and compute a shortest path  $P_j$ . Then select the  $j$  with the most negative paths and among them randomly select a particular  $j'$ . If there are no negative paths, select randomly any  $j'$  with zero path cost if any exists. If there are only positive cost paths stop the procedure, else perform the meeting exchanges indicated by the path and continue. Stop the procedure if the number of consecutive zero cost exchanges exceeds a fixed parameter.

In order to extend the neighborhood size, we have also implemented a mixed local search, denoted LSTP, in which we build at each stage both  $G_j$ , for all parents  $j$ , and  $G_i$  for all teachers  $i$  and select the best chain of moves among all possibilities.

## 6 An Example

Let  $J = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$  and  $I = \{A, B, C, D, E, F\}$  with meeting requests

$$I_1 = \{A, F\}, I_2 = \{B, E\}, I_3 = \{A, B, F\}, I_4 = \{A, D, E, F\}, I_5 = \{A, B\},$$

$$I_6 = \{C, E, F\}, I_7 = \{B, D\}, I_8 = \{C, E, F\}, I_9 = \{C, D, E, F\}.$$

Then we derive

$$J_A = \{1, 3, 4, 5\}, J_B = \{2, 3, 5, 7\}, J_C = \{6, 8, 9\},$$

$$J_D = \{4, 7, 9\}, J_E = \{2, 4, 6, 8, 9\}, J_F = \{1, 3, 4, 6, 8, 9\}.$$

The minimum makespan of the instance is  $T = 6$ , with time slots labeled  $t_1, \dots, t_6$ . By solving the six matching problems with a null objective function we obtain schedule 1 in Figure 1 (rows refer to parents and columns to time slots, the table entries are the teachers and the  $-$  are the idle times), while schedule 2 has been obtained by introducing the objective function (2).

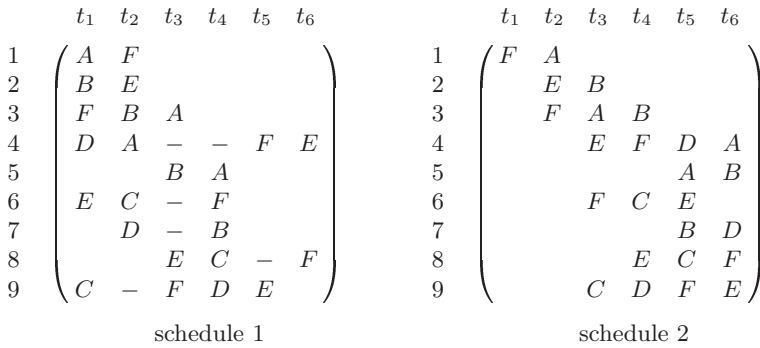


Fig. 1.

As can be seen, the six idle times introduced in the first case are eliminated by taking care of them in the objective function. However, idle times do occur with larger instances even if we use (2) and so we need the local search procedures of Sections 4 and 5. We describe in detail the graph  $G_E$  for schedule 1 and teacher  $E$ , whose meeting sequence is  $(6, 2, 8, -, 9, 4)$ . Let us denote the six nodes of  $G_E$  as  $t_1, \dots, t_6$ . Then  $B_E = \{t_1, t_2, t_3, t_5, t_6\}$  and  $F_E = \{t_4\}$ . At time  $t_1$  teacher  $E$  meets parent 6. This meeting can be moved to time  $t_3$  with the effect of reducing by one idle time, therefore the graph  $G_E$  has the arc  $(t_1, t_3)$  with cost  $-1$ . The meeting can be also moved to  $t_5$  at cost 0 or to  $t_6$  at cost 1. At time  $t_2$  teacher  $E$  meets parent 2. The meeting can be moved to  $t_3, t_4, t_5$  or  $t_6$  at cost 1, 2, 3, 4, respectively. Continuing this way, we build the arcs of graph  $G_E$  whose costs are reported in the following table.

	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$
$t_1$	-	-	-1	-	0	1
$t_2$	-	-	1	2	3	4
$t_3$	2	1	-	-	-1	-
$t_4$	0	0	0	-	0	0
$t_5$	-	-1	-	-	-	1
$t_6$	-	-	-1	-1	-	-

There are some negative cycles in this graph, for instance  $t_1 \rightarrow t_3 \rightarrow t_5 \rightarrow t_6 \rightarrow t_4 \rightarrow t_1$  with cost  $-2$ . Performing the corresponding exchanges leads to schedule 3 in Figure 2. Continuing with teacher  $F$ , we find the cycle  $t_3 \rightarrow t_5 \rightarrow t_3$  of cost  $-1$  and reduce by one more idle time. Then we consider teacher  $C$  and by moving the meeting at  $t_1$  to  $t_3$  we reduce by two more idle times. Finally, we consider teacher  $D$  and move the meeting at  $t_2$  to  $t_3$ , obtaining schedule 4 with no idle times.

As an example of LSP reconsider schedule 1 in Figure 1. The graph  $G_9$  related to parent 9 contains two paths of cost  $-1$ , precisely  $t_1 \rightarrow t_6$  and  $t_1 \rightarrow t_5 \rightarrow t_4$



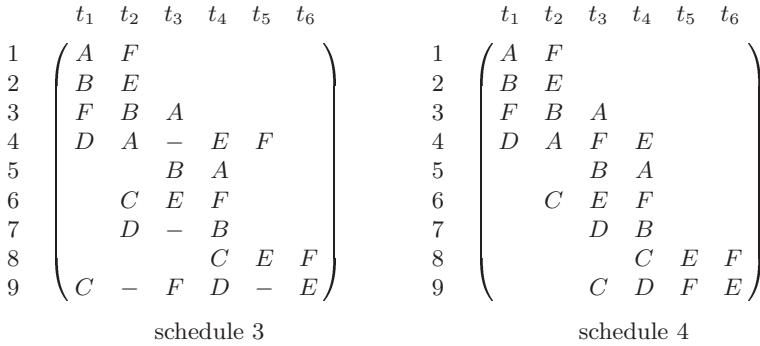


Fig. 2.

$\rightarrow t_6$ . The corresponding sequences of exchanges produce respectively the schedules  $(-, -, F, D, E, C)$  and  $(-, -, F, E, C, D)$ , both having no idle time.

### 7 A Variant of the Problem

In this section we explain how the previous procedure can be adapted to solve an extension of the school meeting problem in which we allow longer meetings for some previously defined parent–teacher pairs. More precisely, we assume that these meetings last twice as long as the normal ones. We do not allow pre-emption of the meetings: i.e., a meeting of double length cannot be interrupted and resumed later.

Under the above conditions, even the problem of finding a schedule of minimum makespan becomes NP-hard. Therefore we do not pursue the objective of finding the minimum makespan, rather we consider it more important to obtain an initial schedule with no pre-emption so that the local search procedure can be designed without worrying about the pre-emption.

Denote by  $p_{ij} \in \{1, 2\}$  the length of the meeting of teacher  $i$  with parent  $j$  and by  $a_i := \sum_j p_{ij}$ ,  $i \in I$ , and  $b_j := \sum_i p_{ij}$ ,  $j \in J$ , the total time required by the meetings of teacher  $i$  and parent  $j$ , respectively. Moreover, set  $T_{\max} := 2 \max \{\max_i |J_i|, \max_j |I_j|\}$  and  $T_{\min} := \max \{\max_i a_i, \max_j b_j\}$ . As one can easily verify, the minimum makespan  $T^*$  of a schedule that assigns all the required meetings without pre-emption is bounded by

$$T_{\min} \leq T^* \leq T_{\max} ,$$

and a schedule with makespan bounded the same way can be easily obtained (just put  $p_{ij} = 2$  for all meetings to obtain a schedule with makespan  $T_{\max}$ ).

In order to solve this variant of the school meeting problem, we modify some aspects of the procedures of Sections 3, 4 and 5. Let us first consider the algorithm of Section 3, which solves a weighted matching problem for each time slot. Since now the makespan  $T$  of a feasible solution is not known in advance, even  $T$  has to be treated as an output of the solution.

Using the same notation as in Section 3, at the beginning of each step  $k$ , three sets  $J_i^{k-1}$ ,  $I_j^{k-1}$  and  $R^{k-1}$  are given, where the set  $R^{k-1}$  contains all the pairs  $(i, j)$  with  $p_{ij} = 2$  whose meeting has started at time  $k - 1$ . Let  $a_i^{k-1}$  and  $b_j^{k-1}$  be the residual times of teacher  $i$  and parent  $j$  at the end of step  $k - 1$  and define  $T^{k-1} := k - 1 + \max \{ \max_i |a_i^{k-1}|, \max_j |b_j^{k-1}| \}$ . Apparently,  $T^{k-1}$  is a lower bound on the makespan of any solution coincident with the current partial schedule on the first  $k - 1$  time slots. We still call critical any parent and teacher having a residual time equal to  $T^{k-1} - (k - 1)$ .

Since pre-emption is not allowed, all the pairs in  $R^{k-1}$  have to be contained in the matching  $M^k$  to be determined at the current stage. In order to guarantee that the matching problem admits a feasible solution, no other rigid constraint on the assignment of selected parents and teachers can be imposed. As a consequence, we have to relax the constraints on the critical teachers and/or parents and take care of them in the objective function. Therefore we assign to each pair  $(i, j)$  a weight of the form

$$w_{ij} := \left\{ \begin{array}{ll} K_1 & \text{if } i \text{ or } j \text{ critical} \\ K_2 & \text{else and if } s_k(j) = 1 \\ -1 & \text{with probability } p_1 \\ 0 & \text{with probability } p_2 \\ 1 & \text{with probability } 1 - p_1 - p_2 \end{array} \right\} \text{ else} \tag{4}$$

where  $K_1 \gg K_2$  to take into account in lexicographic order the two objectives of minimizing the makespan and the number of idle times. At the end of the iteration, the sets  $J_i^k$ ,  $I_j^k$  and  $R^k$  and the values  $a_i^k$ ,  $b_j^k$  and  $T^k$  have to be suitably updated. Note that if at least one critical teacher or parent is not assigned then  $T^k = T^{k-1} + 1$ . The procedure stops when  $a_i^k = 0$  for all  $i$  (or equivalently  $b_j^k = 0$  for all  $j$ ). The final makespan is given by the last value  $T^k$ .

Also the local search procedures described in Sections 4 and 5 have to be slightly modified to avoid the introduction of any pre-emption in the meetings requiring two time slots. To this aim, it is sufficient to change the definition of the arc set of the graph  $G_i$  related to each teacher  $i$  and the graph  $G_j$  related to each parent  $j$  described in those sections. In the case of  $G_i$ , assume that teacher  $i$  meets parent  $j$  at time  $t$ . If the meeting lasts one time slot, then the set of arcs exiting from node  $t$  is defined as before. Otherwise, if the meeting takes two time slots, let say  $t$  and  $t + 1$ , we add at most two arcs: the arc  $(t, t + 2)$  if parent  $j$  is free at time  $t + 2$  and the arc  $(t + 1, t - 1)$  if parent  $j$  is free at time  $t - 1$ . The construction of the arc set of graph  $G_j$  is the same, once the roles of teacher  $i$  and parent  $j$  are exchanged. The cost of the arcs of the two graphs are defined as the number of idle times of the parents that the corresponding move either adds or eliminates from the schedule. We point out that such a move does not introduce any pre-emption in the meeting. Obviously, the number of paths in the graph, and consequently the number of moves, is reduced with respect to the case of meetings of equal duration.

## 8 Computational Results

Let us first consider the main problem with equal time slots. We have tested our procedure on a particular instance given by a local school and also on real-size randomly generated data.

Real data to test our procedure are not actually available, because the proposed advance planning of the meetings has never been thought of. However, we have asked some teachers of a local school to provide reasonable data consistent with their past experience. Today each parent cannot meet more than three or four teachers during a meeting session. It is perceived that asking parents to provide a list of prospected teachers to meet would likely produce long lists requiring too much time to carry out all meetings. Therefore it seems more sensible, in a real implementation, to assign an upper bound on the number of teachers a parent can ask to meet.

With this proviso, we have set up the following data, with 29 teachers and 60 parents and a number of required meetings for each parent between 3 and 6. The instance thus obtained has a makespan of 25 time slots. We have run the assignment phase five times with parameters  $K = 1$ ,  $p_1 = 0.04$ ,  $p_2 = 0.94$ , obtaining solutions with 162, 146, 109, 177 and 122 idle times, respectively. Then for each of these solutions we have separately started LST, LSP and LSTP with 20 as the maximum number of consecutive zero improvement cycles. The local search computations are shown in Figures 3, 4 and 5, where the current idle time number is displayed as a function of the iteration number. We recall that in each iteration a minimum cycle for each teacher has to be computed for LST, a shortest path for each parent for LSP, and both computations have to be carried out for LSTP. The five runs for LST, the five runs for LSP and the five runs for LSTP are shown in Figures 3, 4 and 5, respectively. The ending idle time numbers for the five LST runs are: 4, 0, 3, 3, 2; for the five LSP runs are: 22, 15, 10, 21, 23 and for the five LSTP runs are: 1, 4, 2, 1, 1. The last of these solutions is shown in Figure 6.

As expected, the LST performs much better than the LSP, due to the smaller search space of LSP. In contrast, the increased neighborhood size offered by the two searches in LSTP does not seem to outperform LST. It is only slightly better on average.

An interesting feature is that the improvement per iteration is almost constant in all cases, so that we may roughly say that the number of iterations is almost equal to the initial value of the idle times provided by the assignment phase. In this sense the local search performs well independently of the starting solution. However, the number of iterations is affected by the initial solution and therefore it makes sense to tune the parameters of the assignment procedure in order to start with a good solution.

In the random instances we have fixed the number of teachers and parents to 30 and 80 respectively. For each parent the number of required meetings is a random number  $r \in [r_1, r_2]$ . Then  $r$  teachers are randomly selected to meet

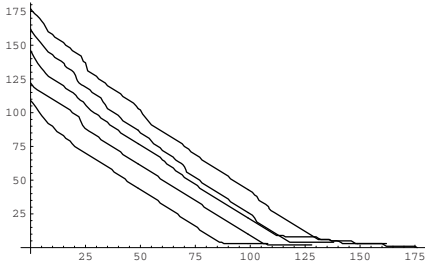


Fig. 3. LST runs

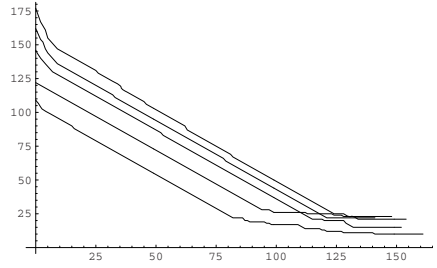


Fig. 4. LSP runs

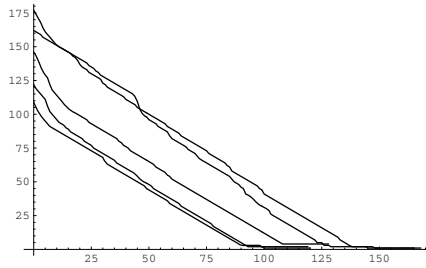


Fig. 5. LSTP runs

that parent. We have generated six instances. For the instances 1 and 2 we have used  $r \in [2, 4]$  (*sparse* instances); for the instances 3 and 4 we have used  $r \in [4, 6]$  (*normal* instances) and for the instances 5 and 6 we have used  $r \in [4, 8]$  (*dense* instances). For each instance we have run the assignment phase four times (with parameters as before) and for each solution of the assignment problem we have separately started LST, LSP and LSTP. The computational results are shown in Tables 1, 2 and 3, where columns (1) report the instance number (in boldface) and its makespan, columns (2) report the idle time number at the end of the assignment phase, and, for each local search, columns (3) and (4) report the final idle time numbers and the iteration numbers respectively. The four rows for each instance refer to the four computations.

As in the previous case the tests show that LST performs much better than LSP. Again, there is no clear winner between LST and LSTP. As a strategy to obtain good solutions, we suggest running both procedures several times. We have no explanation as to why LSTP is not definitely better than LST.

We have also tested the variant of Section 7 on instances with 15 teachers and 60 parents and a random number of meetings for each parent in  $[3, 6]$ . The only difference is that each meeting is randomly assigned duration 1 or 2 with probability 0.8 and 0.2, respectively. The matching phase has been carried out with weights given by (4) ( $K_1 = 10^4$ ,  $K_2 = 10$ ,  $p_1 = 0.04$ ,  $p_2 = 0.94$ ). The makespan, the number of idle times at the end of the matching phase and at the end of the modified local search procedure are for the 5 runs 30, 71 and 11; 29,

1	16 26 11 1 21 28				
2		11 21 1			
3	28 21 16 11 26				
4					16 21 28 26
5	11 1 28 16 21 26				
6	28 21 11 1				
7				21 16 11 26 28	
8		20 22 11 27 29			
9			22 20 27 29 2		
10		27 20 22			
11			11 20 22 2 27 29		
12	2 20 27 22				
13				22 2 27 29	
14	12 26 17				
15		12 17 3 24 28 26			
16	26 28 17 3 12 24				
17					24 12 17
18		3 17 12 26 28 24			
19		3 17 24 28 12			
20	13 4 19				
21	27 13 19 28				
22		4 27 13 23 28 19			
23			4 28 27 23		
24			4 13 19 23 28 27		
25					14 26 29
26		23 14 18			
27	29 5 23 14				
28		5 14 26 18 23 29			
29	29 5 18 26 23 14				
30				18 5 26 29	
31		14 18 5 23 26 29			
32				6 29 26	
33	29 24 13 6 19 26				
34		13 19 29			
35			19 13 26 29 6 24		
36	19 29 24 6				
37				6 13 29 19 24	
38	24 6 13 29				
39	22 7 16 27 29				
40	27 7 14 16 22				
41				16 22 7 29 14 27	
42	27 7 14				
43		14 22 29 27			
44		27 16 29 22 14			
45	15 21 20 8 26				
46			21 20 15 26 8 28		
47		20 8 28 15			
48			15 8 20 21		
49		21 20 15 26 8 28			
50	9 - 12 27 28 25				
51				17 25 27 28	
52		9 12 25 17 28 27			
53				12 17 25	
54	25 17 9 28 12 27				
55		17 25 9 12 28 27			
56	10 18 25 15 29 27				
57		18 10 29			
58		15 25 18 10 29			
59	25 15 10				
60			18 15 29 27		

Fig. 6. Solution with one idle time

**Table 1.** Sparse instances

		LST		LSP		LSTP	
(1)	(2)	(3)	(4)	(3)	(4)	(3)	(4)
	25	1	39	3	102	2	37
<b>1</b>	13	2	32	5	29	2	31
14	25	2	55	6	67	0	40
	16	2	69	6	59	0	41
	18	1	58	2	50	1	37
<b>2</b>	18	2	46	7	40	0	25
12	27	1	57	7	65	0	29
	25	3	39	9	34	6	35

**Table 2.** Normal instances

		LST		LSP		LSTP	
(1)	(2)	(3)	(4)	(3)	(4)	(3)	(4)
	71	6	71	9	114	8	69
<b>3</b>	58	1	75	23	71	3	63
18	51	8	73	26	58	9	81
	41	1	78	9	101	2	66
	68	3	118	12	95	2	111
<b>4</b>	54	2	95	10	99	3	98
21	70	1	116	15	130	2	86
	63	1	95	15	86	2	89

**Table 3.** Dense instances

		LST		LSP		LSTP	
(1)	(2)	(3)	(4)	(3)	(4)	(3)	(4)
	112	8	136	40	133	5	158
<b>5</b>	131	8	166	38	186	13	110
27	75	7	116	29	89	7	101
	133	8	141	25	186	7	163
	73	6	140	16	172	12	92
<b>6</b>	110	6	170	14	251	6	167
22	93	10	88	14	184	5	175
	164	14	143	23	254	5	213

73 and 21; 30, 76 and 16; 30, 73 and 20; 30, 71 and 11. As anticipated, the local search is not as effective as in the normal case.

## References

1. Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, 75–102 (2002)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, NJ (1993)
3. Bartholdi, J.J, McCroan, K.L.: Scheduling interviews for a job fair. *Operations Research* 38, 951–960 (1990)
4. Dowsland, K.A.: Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research* 106, 393–407 (1998)
5. Giaro, K., Kubale, M.: Compact scheduling of zero-one time operations in multi-stage system. *Discrete and Applied Mathematics* 145, 95–103 (2004)
6. Gonzalez, T.: Unit execution time shop problems. *Mathematics of Operations Research* 7, 57–66 (1982)
7. Gonzalez, T., Sahni, S.: Open shop scheduling to minimize finish time. *Journal of the ACM* 23, 665–679 (1976)
8. Hall, N.G., Srisankarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research* 44, 510–525 (1996)
9. Meyers, C., Orlin, J.B.: Very large-scale neighborhood search techniques in timetabling problems. In: Burke, E.K., Rudová, H. (eds.) *PATAT 2006*. LNCS, vol. 3867, pp. 24–39. Springer, Heidelberg (2007)

# Hierarchical Timetable Construction

Jeffrey H. Kingston

School of Information Technologies,  
The University of Sydney, NSW 2006, Australia  
jeff@it.usyd.edu.au  
<http://www.it.usyd.edu.au/~jeff>

**Abstract.** A hierarchical timetable is one made by recursively joining smaller timetables together into larger ones. Hierarchical timetables exhibit a desirable regularity of structure, at the cost of some limitation of choice in construction. This paper describes a method of specifying hierarchical timetables using mathematical operators, and introduces a data structure which supports the efficient and flexible construction of timetables specified in this way. The approach has been implemented in KTS, a web-based high school timetabling system created by the author.

## 1 Introduction

The basic timetable construction problem is to assign times and resources (students, teachers, rooms, etc.) to a set of meetings so that the resources have as few timetable clashes as possible. To this basic problem many other constraints are typically added, such as that the times allocated to a meeting be spread evenly through the week, that workload limits placed on some resources not be exceeded, and so on. Timetable construction is an NP-complete problem with an extensive literature [4,5,6,7,8].

Informally, a *regular timetable* is one in which a pattern may be discerned which makes the timetable easy to understand and remember. Regularity may take many forms, but this paper will be chiefly concerned with regularity in the choice of times. For example, North American universities commonly require all courses to occupy three hours per week, offered in one of the sets of time slots Mon–Wed–Fri 9–10AM, or Mon–Wed–Fri 10–11AM, and so on, producing a very regular timetable.

Even when such a strict rule as this is not possible, still some regularity might be achievable, perhaps by attempting to minimize the number of pairs of meetings that share at least one time, in addition to the usual objectives.

Regular timetables are easy to assign resources to. For example, in the North American university system, each meeting can meet in the same room for all three of its times. This point is particularly significant in high school timetabling, the area of timetabling which has inspired this paper, since teachers are assigned as well as rooms. Teacher assignment is the main area where the author's previous work in high school timetabling [10,13] is deficient. Thus, regularity is more than just an aesthetic consideration.



This paper introduces a method of specifying regular timetables hierarchically, using *timetable expressions* analogous to algebraic expressions. This seems to be the first paper to find a use for arbitrarily deep hierarchies, although two-level hierarchies have been used occasionally. For example, Fizzano and Swanson [11] group together pairs of meetings, where one occurs on Mon–Wed–Fri and the other on Tue–Thu, and Adriaen et al. [1] aggregate sets of university meetings that occur in disjoint weeks of the semester.

As will be seen, the particular kind of hierarchical specification introduced in this paper imposes significant hard constraints on the times assigned to the meetings of the hierarchy. This is appropriate for a method used in high school timetabling, where the constraints tend to be relatively hard because entire classes of students suffer under any deficiencies, rather than individual students as in, for example, examination timetabling.

One way to handle these constraints would be to express them in a general-purpose constraint programming language. A number of papers have taken this approach [9,16]. However, this paper takes the special-purpose route, introducing a data structure, the *layer tree*, which represents timetable expressions and efficiently supports a particular set of constraints relevant to timetabling. The special-purpose route, while costly in development time, has some advantages. In particular, some of the algorithms used here, for example weighted bipartite matching, do not seem to be available in any existing constraint programming system [3,17], although some recent research into the *all-different* constraint [12], which implements unweighted bipartite matching, is a step in that direction.

Our focus is on the efficient implementation of some basic assignment and deassignment operations (and the resulting constraint propagation), rather than their use with any particular timetable construction algorithm. If these operations are efficient, many algorithms, including construction heuristics, tree searches, and local searches, become available. Although efficiency is a key goal, it has not been considered useful to report running times, since the operations to be presented are all polynomial time, and running times say more about the algorithms built on these operations than the operations themselves.

Layer trees are particularly effective when sets of meetings can be identified that must be disjoint in time. In high school timetabling, each set of meetings attended by a given student group satisfies this condition. This author's KTS timetabling system [14], a free, public web site for high school timetabling, uses layer trees. KTS typically produces a good timetable in about ten seconds [15], showing that layer trees can support practical timetabling.

The algorithms used here have appeared in previous timetabling work by the author and others [10,13,18]. This paper's contribution is to show how these algorithms can be incorporated into a flexible, efficient, hierarchical constraint framework. Section 2 introduces timetable expressions, and Section 3 introduces the layer tree data structure. Section 4 analyses the problem of efficiently propagating constraints related to time through this data structure as assignments and deassignments occur, and Section 5 does the same for resource constraints. Section 6 surveys some other, less fundamental features implemented in KTS.

## 2 Timetable Expressions

The idea of using an expression to specify a problem is well known in logic. Consider a Boolean expression such as

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3).$$

The expression defines an instance of the satisfiability problem, for which a solution consists of an assignment of values to the variables which satisfies the expression. In the same way, *timetable expressions* will be used to specify timetable construction problems.

The simplest kind of timetable expression is the *time variable*, a variable  $v$  whose domain is some subset of the set of available times  $T$ . This domain may change as solving proceeds; its value at some moment will be denoted  $tdom(v)$ , and its initial value, specified when the variable is created, will be denoted  $tdom_0(v)$ . For example, if  $v$  may be assigned any time, then  $tdom_0(v) = T$ ; if  $v$  is *preassigned* to a specific time  $t$ ,  $tdom_0(v) = \{t\}$ . Other initial domains may constrain times to be during the mornings, or on Mondays, and so on.

The ultimate aim is to assign an element of  $T$  to every time variable, just as the aim is to assign a Boolean value to every variable when solving satisfiability problems. However, it turns out that in hierarchical timetabling a more useful basic operation is the assignment of one time variable,  $v$ , to another,  $w$ , with the meaning that  $v$ 's value is constrained to be equal to  $w$ 's. Assigning one variable to another expresses the idea that two meetings are to occur simultaneously, without having to say when.

Thus, our system offers two basic operations: assigning one variable  $v$  to another variable  $w$ , and removing the assignment of  $v$  to  $w$ . A variable may be assigned to at most one other variable at any moment; but that other variable is free to be assigned to a third variable (or not), and many variables may be assigned simultaneously to one variable.

Two timetable expressions  $e_1$  and  $e_2$  may be joined using the *concatenation operator*, written  $e_1e_2$ , meaning that the times assigned to the variables of  $e_1$  must be disjoint from the times assigned to the variables of  $e_2$ . For example, a meeting requesting four times may be expressed by the timetable expression  $v_1v_2v_3v_4$ , where  $v_1$ ,  $v_2$ ,  $v_3$ , and  $v_4$  are time variables. Concatenation specifies that the times assigned to these four variables must be distinct, as required, in addition to any constraints on their domains.

If two meetings request the same resource, and it is a hard constraint that that resource may have no clashes in its timetable, then the expressions representing those two meetings may be concatenated. This is fundamental in the high school timetabling work which motivates this paper: each student group is such a resource, and the meetings it appears in must be disjoint in time.

Two timetable expressions  $e_1$  and  $e_2$  may be joined using the *alternation operator*, written  $e_1 + e_2$ , meaning that  $e_1$  and  $e_2$  are to appear in the same timetable, but there are no time constraints between their variables. In the high school timetabling application,  $e_1$  might represent the meetings attended

by one student group, and  $e_2$  might represent the meetings attended by some other student group. These two sets of meetings have no time interdependencies, so joining them with  $+$  is appropriate. If there is a meeting that both student groups attend, then its expression ( $v_1v_2v_3v_4$  or whatever) will appear in both subexpressions, and its variables must be assigned times disjoint from those assigned to the variables its expression is concatenated with in both subexpressions.

These operations are named by analogy with the corresponding operators of regular expressions:  $e_1+e_2$  signifies that  $e_1$  and  $e_2$  are alternative activities, while  $e_1e_2$  signifies that one activity must follow after the other. In timetable expressions, however, both operators are associative and commutative. A distributive law,  $(a + b)c = (ac + bc)$ , also holds.

Finally, there is the *restriction operator*, written

$$w_1w_2 \dots w_k : e$$

where  $w_1w_2 \dots w_k$  is a concatenation of time variables called *restriction variables*, and  $e$  is a timetable expression. This specifies that each variable in  $e$  must not appear outside  $e$ , and must be assigned to one of the  $w_i$  (which themselves must be assigned disjoint times), restricting  $e$  to a timetable using at most  $k$  times.

Restriction introduces abstraction into timetable expressions. The expression  $e$  may be timetabled into  $w_1w_2 \dots w_k$  independently of the rest of the problem, after which these variables are indistinguishable from an ordinary concatenation of variables describing a meeting.

Typically, the outermost level of a timetable expression is a restriction expression which limits the timetable to the available times. Letting  $T = \{t_1, t_2, \dots, t_n\}$  be the set of available times, this expression would have the form

$$w_1w_2 \dots w_n : e$$

where  $tdom_0(w_i) = \{t_i\}$  for all  $i$ . The operation of assigning a particular time  $t_i$  to a variable  $v$  is not offered, but assigning  $v$  to  $w_i$  is effectively the same thing.

To *solve* a timetable expression is to assign each variable  $v$  to one of the restriction variables  $w_i$  in its nearest enclosing restriction expression, without violating the constraints given above; variables not so enclosed remain unassigned. An example of a small timetable expression and its solution appears in Figure 1.

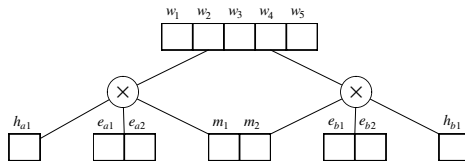
Variants of the timetabling problem exist in which the exact number of available times is not given; instead, a timetable with as few times as possible is sought, consistent with other requirements. The restriction notation could easily be extended to cover such problems. However, the algorithms appearing later in this paper assume a fixed number of variables, so any such ‘extensible restriction’ would have to be solved (or at least, its number of variables determined) before incorporation into a larger timetable, forcing a bottom-up solution order.

	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$
7A	7AB-Mathematics		7A-Hist	7A-English	
7B			7B-English		7B-Hist

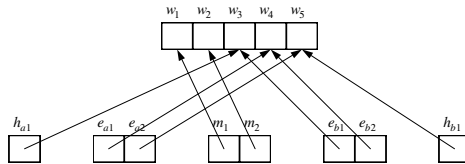
(a) A small timetable, or *tile*, occupying two student groups (7A and 7B) for five times  $t_1, t_2, t_3, t_4,$  and  $t_5$ .

$$w_1w_2w_3w_4w_5 : m_1m_2h_{a1}e_{a1}e_{a2} + m_1m_2e_{b1}e_{b2}h_{b1}$$

(b) A timetable expression for which (a) is a solution. Here  $w_1w_2w_3w_4w_5$  represent the five available times,  $h_{a1}$  represents 7A-Hist,  $e_{a1}e_{a2}$  represents 7A-English, and so on;  $m_1m_2$ , representing 7AB-Mathematics, lies in two subexpressions.



(c) A layer tree corresponding to (b). Variables are shown as labelled boxes; + nodes are shown as concatenations of their variables.



(d) The layer tree of (c), showing assignments representing the timetable of (a). The  $\times$  nodes have been omitted for clarity.

**Fig. 1.** Timetables, timetable expressions, layer trees, and assignment

### 3 The Layer Tree Data Structure

A timetable expression such as

$$(e_1 + e_2)(e_3 + e_4)$$

is difficult to handle, since it is not clear how many of the available times should be allocated to  $e_1 + e_2$ , and how many to  $e_3 + e_4$ . While cases of this kind cannot be ruled out entirely, they seem not to occur in high school timetabling, and they are not supported by KTS; so we proceed now to exclude them.

A *simple timetable expression* is one in which each alternation expression  $e_1 + \dots + e_m$  is immediately enclosed in a restriction expression. In such expressions it is easy to determine how many times to allocate to each subexpression.

Furthermore, a simple timetable expression can be analysed into a tree (or forest if the root is a concatenation expression) of expressions of the form

$$w_1 w_2 \dots w_n : (e_{11} e_{12} \dots e_{1k_1} + \dots + e_{m1} e_{m2} \dots e_{mk_m})$$

called a *restricted sum of products*. Here  $m$  may be 0, in which case the expression just denotes a sequence of variables  $w_1 w_2 \dots w_n$ . Each  $e_{ij}$  is a restricted sum of products. Some of the  $e_{ij}$  may be shared, i.e. some  $e_{pq}$  and  $e_{rs}$  may be the same subexpression. To *solve* a restricted sum of products is to assign each of the restriction variables in each  $e_{ij}$  to one of the  $w_i$ .

One way to solve a timetabling problem represented by a simple timetable expression is to solve its restricted sums of products in bottom-up order. This paper aims for more flexibility, however, in allowing assignments and deassignments within each restricted sum of products at any moment. For example, this would permit the timetable of a small component to be adjusted (by local search, perhaps) after that component is incorporated into a larger timetable. To achieve this we need a data structure which represents the entire tree of restricted sums of products, with the current state of the assignments of each.

The data structure we will use, which we call a *layer tree*, is essentially just the expression tree corresponding to a simple timetable expression. A layer tree has two types of nodes:  $+$  nodes representing restricted sums of products and containing their restriction variables, and  $\times$  nodes representing concatenations. Nodes of both types may have any number of children. Figure 1 gives an example of converting a restricted sum of products into a layer tree.

Without loss of generality, we may assume that in every layer tree the root is a  $+$  node, its children are  $\times$  nodes, their children are  $+$  nodes, and so on, with the node type alternating between  $+$  and  $\times$  at each level. To bring an arbitrary layer tree into this form, first use the associativity of concatenation to replace every  $\times$  node whose parent is a  $\times$  node by its children. Then insert a  $\times$  node immediately above every  $+$  node whose parent is a  $+$  node. Finally, if the root is a  $\times$  node, remove it and solve each of its children independently.

Each variable  $v$  within each  $+$  node other than the root node requires assignment to a variable  $w$  in the  $+$  node two levels above it. Each such assignment is represented by a pointer in  $v$  to  $w$  (Figure 1(d)). Eventually, when all these variables are assigned in this way, every variable may be said to have been assigned a time, obtainable by following the chain of pointers to its end. This arrangement is essentially that used when unifying variables in logic programming.

Any set of variables requiring distinct times is called a *layer*. The variables lying in any  $+$  node form a layer; the variables lying in all the children of any  $\times$  node also form a layer.

For example, the author's KTS system builds a layer tree with several levels. Each meeting may contain submeetings which have to be timetabled into the times of the meeting; each such meeting becomes a restricted sum of products. Then small groups of compatible meetings are timetabled together, producing *tiles* such as the one in Figure 1(a); each tile is the solution of a restricted sum of products whose child layers contain meetings. Finally, the times of the week form a restricted sum of products whose child layers contain tiles.

## 4 Time Constraints

This section explains how constraints on time assignment are propagated through the layer tree, so that at any moment it is clear for each variable exactly which variables it may be assigned to without violating any time constraints.

Since each variable is assigned to at most one other variable at any moment, the assignments form a directed forest with edges pointing towards the roots. The current assignment of a variable  $v$  will be denoted  $p(v)$  ('parent of  $v$ ') when present, and the variable at the root of the tree of assignments containing  $v$  (possibly  $v$  itself) will be denoted  $r(v)$ . A *root variable* is a variable  $w$  such that  $r(w) = w$ . Every variable in the root node of a layer tree must be a root variable, but other variables may also be root variables: root variables are just variables that are currently not assigned to other variables.

Recall that each time variable  $v$  has its *initial domain*  $tdom_0(v)$  of times that it may be assigned initially, and its *current domain*  $tdom(v)$  of times that it may be assigned to at the current moment. We require

$$tdom(v) \subseteq tdom_0(v)$$

since otherwise the original constraint has been lost.

Each time variable  $v$  has a second kind of domain, its *variable domain*  $vdom(v)$ , which is the set of variables that  $v$  may be assigned to. Again,  $vdom_0(v)$  will denote the initial value of  $vdom(v)$ , and we require  $vdom(v) \subseteq vdom_0(v)$ . For each variable  $v_{ij}$  in the restricted sum of products

$$w_1 w_2 \dots w_m : (v_{11} v_{12} \dots v_{1k_1} + \dots + v_{m1} v_{m2} \dots v_{mk_m})$$

we have  $vdom_0(v_{ij}) \subseteq \{w_1, w_2, \dots, w_m\}$ .

The two domains are related by the condition

$$w \in vdom(v) \Rightarrow tdom(w) \subseteq tdom(v)$$

( $\Rightarrow$  is implication). For example, this prohibits a preassigned variable from being assigned to an unpreassigned one; in general, it prevents  $w$  from being assigned a time not acceptable to  $v$ .

The following formulas show how  $tdom(v)$  and  $vdom(v)$  may be kept up to date as variables are assigned and deassigned:

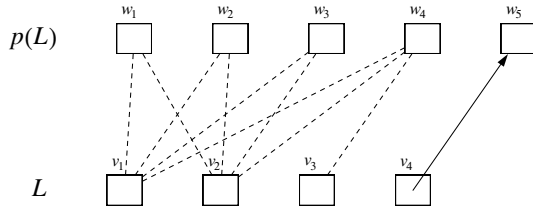
$$tdom(v) = tdom_0(r(v))$$

and

$$vdom(v) = \{w \in vdom_0(v) \mid tdom(w) \subseteq tdom(v)\}.$$

These follow easily from the discussion so far. Note that  $vdom(v)$  is only needed at moments when  $v$  is not assigned.

When a variable  $v$  is assigned to a variable  $w$ , the variable domains of all variables concatenated with  $v$  need to be reduced by removing  $w$ , since assigning any of them to  $w$  would violate the constraint that concatenated variables must be assigned distinct times. An efficient method of doing this is as follows.



**Fig. 2.** An example of an unweighted bipartite matching graph between the variables of a child layer  $L$  and its parent layer  $p(L)$ , shown as dashed edges. One assignment is already present, from  $v_4$  to  $w_5$ , ensuring that  $L \in pl(w_5)$  and thus excluding  $w_5$  from  $vdom(v)$  for all other  $v \in L$ . This particular matching could arise when  $v_3$  and  $w_4$  are preassigned to the same time ( $tdom_0(v_3) = tdom_0(w_4) = \{t_i\}$  for some  $t_i \in T$ ), and the other variables are free to be assigned any time. Note that  $w_4 \in vdom(v_1)$  but no maximal matching would assign  $v_1$  to  $w_4$ .

Let the set of variables lying in the children of one  $\times$  node be  $v_1, \dots, v_m$ ; these variables form a layer which we call  $L$ . The variables  $w_1, \dots, w_n$  in the parent of that  $\times$  node form another layer, which we call  $p(L)$ . The variables of  $L$  must be assigned to the variables of  $p(L)$ .

For each  $v_j$ , define the *child layer set*,  $cl(v_j)$ , to be the set of  $\times$  nodes which are the parents of the  $+$  node containing  $v_j$ . (As explained earlier, a  $+$  node may have several parents, typically because the meeting it represents contains several preassigned resources.) For each  $w_i$ , define the *parent layer set*,  $pl(w_i)$ , to be the union of the child layer sets of all variables assigned directly to  $w_i$ . Parent layer sets must be maintained dynamically as assignments are done and undone.

Now modify the definition of  $vdom(v)$  given above to

$$vdom(v) = \{w \in vdom_0(v) \mid (tdom(w) \subseteq tdom(v)) \wedge (cl(v) \cap pl(w) = \emptyset)\}.$$

This excludes  $w$  from  $vdom(v)$  when some other variable that shares a layer with  $v$  is currently assigned to  $w$ . The set operations may be implemented efficiently using bit vectors.

Given current values of  $vdom(v)$  for all variables  $v$  in some layer  $L$ , the next question is whether it is possible to assign all the currently unassigned variables of  $L$  to variables in  $p(L)$ . Since the assignments must be to distinct variables, this is an unweighted bipartite matching problem between the currently unassigned variables of  $L$  and the variables of  $p(L)$ , with edges defined by the domains  $vdom(v)$  of the currently unassigned variables of  $L$  (Figure 2). We will see in the next section that there are reasons for preferring some assignments to others, converting the unweighted bipartite matching into a weighted one.

## 5 Resource Constraints

In addition to requests for times, meetings contain requests for resources. These may be for particular resources, called *preassigned resources*, or for any resource of a certain type, such as a Science laboratory.

A typical high school meeting would request one preassigned student group resource, one teacher resource which may or may not be preassigned, and one room, usually not preassigned. However, it is very common for a whole collection of meetings to be required to run simultaneously, to give the students a choice of activities. Such a collection would be modelled as a single large meeting with many resource requests.

A basic question which can be asked of any set of meetings is whether the institution has sufficient resources to allow those meetings to run simultaneously. For example, if a school has only two Music teachers and two Music rooms, then at most two Music meetings may run simultaneously. As is well known, this question can be answered using an unweighted bipartite matching model, called a *resource sufficiency matching* [10], as follows.

For each request for a resource in each of the meetings involved, create one node called a *demand node*. For each resource in the instance of the timetabling problem being solved, create one node called a *supply node*. Connect each demand node to those supply nodes capable of satisfying that demand. For example, a demand node for a particular student group resource would be connected to just the supply node representing that resource; a demand node for a Science laboratory would be connected to every supply node representing a Science laboratory. The meetings may run simultaneously if a maximum matching in this graph touches every demand node. The matching defines an assignment of resources to requests which satisfies as many requests as possible.

This model allows supply nodes which are capable of satisfying several kinds of demands: teachers who teach both English and History, rooms which are Science laboratories but are usable as ordinary classrooms, and so on. The obvious simpler method, of comparing the total number of demands of each type with the total supply of resources of that type, fails to handle such cases.

We turn now to the implementation of these ideas within the layer tree. Associated with each time variable is a set of demand nodes, which we call a *demand chunk*. For example, a Music meeting might request student group  $\mathcal{7C}$ , one Music teacher, and one Music room for four times, and then there will be four variables, each with an associated chunk containing three demand nodes. These chunks happen to be identical, but they are copies, not shared.

Any time variable may have a demand chunk, whether or not it derives from a meeting. In high school timetabling, for example, the variables of the root layer have chunks that express resource unavailability: if a resource  $r$  is unavailable at time  $t_i$ , then the chunk associated with root layer variable  $w_i$  will contain a demand for  $r$ .

The layer tree treats time constraints as hard constraints, in that it is not designed to track the number of violations of these constraints, merely to prohibit them. For resource constraints however we have a free choice of whether to treat them as hard or soft constraints, and we will follow the KTS implementation in treating them as soft constraints. The aim is therefore not to fail when resources are insufficient, but rather to report the number of unmatchable demand nodes. This is calculated by having one bipartite graph for each root variable, in which all the demand chunks of all the variables assigned to that root variable directly



or indirectly are accumulated (since the assignments have caused these demands to be simultaneous), and supply nodes for all the resources of the instance as usual, and finding a maximum matching in each of these graphs.

The standard algorithm for unweighted bipartite matching has some useful properties which permit matchings to be calculated in an incremental manner. Briefly, one can push and pop demand chunks onto and off a matching graph in stack order (last-in-first-out) without recalculating the matching from scratch. The supply nodes remain constant throughout. Thus, when an assignment of  $v$  to  $w$  is made, one can simply push the demand chunks from  $v$ 's subtree (that is, the chunks associated with  $v$  and every variable currently assigned to  $v$ , directly or indirectly) onto  $r(w)$ 's matching graph; when a deassignment of  $v$  to  $w$  is made, one must pop chunks off  $r(w)$ 's graph until all  $v$ 's subtree's chunks are popped, then push back onto  $r(w)$ 's graph all chunks that were popped off during this process that were not from  $v$ 's subtree. The KTS implementation uses lazy evaluation, merely recording requests for pushes and pops, and not doing anything until a request for the number of unmatched nodes is received, at which point one sequence of pops followed by one sequence of pushes brings the matching up to date.

We return now to the unweighted bipartite matching problem mentioned at the end of the preceding section, between the unassigned variables of a layer  $L$  and the variables of its parent layer  $p(L)$ . For each unassigned variable  $v$  of  $L$ , we saw that the current domain  $vdom(v)$  determines which edges to place in the bipartite graph. Now with each such edge, from  $v$  to  $w$  say, we can associate a cost: the number of additional unmatched nodes that would occur if  $v$  was assigned to  $w$ , calculated by matching the chunks of  $v$ 's subtree and  $r(w)$ 's subtree together without actually making the assignment. A maximum matching between the unassigned variables of  $L$  and  $p(L)$  of minimum total cost will give a lower bound on the number of additional unmatched demand nodes that will occur when the unassigned variables of  $L$  are assigned to variables of  $p(L)$ . This model has been called *weighted meta-matching* in [13], where it provides a valuable forward check.

The KTS implementation recalculates edge costs only when changes to the demands at either end make that necessary. It calculates weighted matchings lazily on demand, but not incrementally. Although a well-known algorithm exists which can do this, by finding negative-cost cycles in the residual graph, it is slow since it requires the use of the Bellman–Ford shortest path algorithm rather than Dijkstra's algorithm [2]. Fortunately the graphs are small, since the number of nodes per layer is at most the number of times in the cycle (typically about 40 for a one-week cycle, or 60 for a two-week cycle), so calculating these weighted matchings from scratch is not time consuming.

## 6 Other Features

In this section we briefly survey some other features of the KTS layer tree. They serve as examples of how the basic ideas can be extended.

**Time blocks.** A sequence of times that follow each other chronologically without a break is called a *time block*. For example, the first four times on Monday might form a time block. Then after a lunch break there might be four more times followed by an end-of-day break. In KTS, meetings may request that their times have a particular *block structure*. For example, a meeting with 6 times might request two doubles (blocks of two times) and two singles.

The KTS layer tree allows time variables to be grouped into blocks. The time variables of a layer of meetings are grouped into blocks defined by the meetings' block structure requests; the time variables of the root layer (representing the times of the week) are grouped into blocks representing the sets of times between the naturally occurring breaks.

An initial problem is to determine whether the time blocks of some layer can be packed into the time blocks of the week, allowing for the fact that (for example) a block of four times on Monday morning can be split into two doubles, or one double and two singles, or whatever is required. This is an NP-complete bin packing problem, but real instances are small and easily solved.

Once such a packing has been found, and the large blocks of the week broken down into smaller blocks that exactly match the meetings' block structure requests, the layer tree implements a weighted meta-matching between blocks rather than individual variables. Two blocks are connected by an edge if they have the same number of variables and corresponding variables within the blocks would be connected by an edge in the unblocked matching. The cost of a block-to-block edge is the sum of the costs of the variable-to-variable edges it replaces.

The layer tree offers a heuristic algorithm which simultaneously carries out the bin packing and builds the blocked matching. Initially the variables are all in separate blocks, producing the unblocked matching described earlier. This matching must touch every child block, otherwise the algorithm would have already failed. One by one in decreasing width order, the child blocks are introduced by merging their unblocked variables into a block. If this causes the matching to fail to touch every child block (as it will whenever the child block has width greater than one), an attempt is made to remedy this by merging unblocked parent variables into blocks of the appropriate size. All possible mergings are tried, and the best of these as measured by the cost of the resulting matching is kept; or if no merging exists, then as a last resort, one child block (usually the one just introduced) is split up again; the solution will have a defective block structure at that point.

The decisions about how to split parent blocks made by this algorithm depend on the state of resource sufficiency in those blocks' variables. Consequently it is not useful to build a blocked matching for every child layer of a restricted sum initially. Rather, the usual unblocked matchings are built for each child layer, then a child layer's unblocked matching is replaced by a blocked matching as the first step in assigning that layer. The blocked matching is a temporary structure, only in existence while its layer is being assigned.

Blocked matchings suffer from an awkward problem. Suppose a meeting requires one double and one single block. The matching assigns the double to the

first two times on Monday; it assigns the single to the third time on Monday. The result is a triple, not a double plus a single. Finding a minimum-cost matching which avoids this problem appears to be NP-complete. KTS's weighted meta-matching algorithm discourages such assignments by artificially increasing the cost of augmenting paths that would produce them. The implementation has been done with care, and runs in time which is often a small constant, and at worst is proportional to the length of the augmenting path being considered. The idea is purely heuristic, to be sure, but it seems to work well.

Many other conditions besides time blocks may be imposed on sets of times. A meeting's times may be required to be spread evenly through the week, the times of the meetings attended by a student group may be required to be *compact* (contain no gaps within any day), and so on. The author has not yet attempted to support such conditions within the layer tree.

**Regularity.** The layer tree supports regularity by supporting hierarchical timetable construction, but this does not of itself encourage regularity between the child layers of each  $+$  node. We mentioned earlier a straightforward way to do this, by partitioning the variables of the parent layer into sets, called *columns*, whose size is a typical meeting size, and assigning meetings to entire columns wherever possible. This was the North American universities' approach.

Columns are supported by the layer tree by allowing temporary reductions in  $vdom(v)$ . An algorithm might restrict the domains of the variables of a meeting to one column, then check the total resource sufficiency badness of the entire layer tree; if it has not increased, assigning that meeting to that column may be good. The layer tree also maintains, for each set of variables representing one meeting, a count of the number of distinct columns that that meeting's variables are assigned to. The total of all these counts measures the current irregularity.

**Evenness.** It is desirable for demand for a particular type of resource to be spread evenly across the week, not concentrated at particular times. This is because resource assignment struggles at times when every resource of a particular type is required: there are enough resources, perhaps, but there is little freedom of choice. This property we call *evenness*.

Evenness, like resource sufficiency, depends on the resource demands made at each time, so the layer tree's support for it is very similar to its support for resource sufficiency. (There does not seem to be any efficient way to extract evenness information from the resource sufficiency matchings themselves.) The total demand for each type of resource is maintained in root variables. The sum of the squares of these totals is an effective and easily updated overall measure of unevenness. For example, two root variables each demanding a quantity  $a$  of some type of resource contribute  $2a^2$  to total unevenness. If the timetable is changed so that one demands quantity  $a - 1$  and the other demands  $a + 1$ , these less even demands contribute  $2a^2 + 2$  to unevenness. Demands from the same faculty (e.g. Junior English and Senior English) are considered to be the same type of demand, since they typically have many resources in common.

**Overall badness.** For the convenience of algorithms that use the layer tree, the KTS implementation offers access to an object holding the current total badness of the tree, as a triple whose first component is the number of resource sufficiency defects implied by the current state (the total number of unmatched nodes in resource sufficiency matchings, plus the total cost of all meta-matchings), and whose second and third components are the irregularity and unevenness, measured as just described. Each data structure responsible for calculating any badness value at any point in the tree also takes responsibility for reporting any change to this global badness object, or at least reporting itself as out of date and needing recalculation the next time a badness value is requested.

## 7 Conclusion

This paper has defined a form of hierarchical timetable specification and shown how support for it can be implemented efficiently using the layer tree data structure. Time assignments and deassignments may be carried out at any point in the tree, and an efficient constraint propagation algorithm updates the domains of the variables and reports the consequences for resource sufficiency at each time. Extensions to the basic framework, supporting block structure, regularity, and evenness, have been implemented in the author's KTS system.

Despite these successes we must acknowledge that the methods used here are very specific, limiting their wider application. For example, one constraint type, familiar from university timetabling, requires that one meeting occur earlier in the cycle than another. Although one could incorporate such constraints easily enough (including the requisite updates of variable domains), if there are many of them the value of the lower bounds provided by the minimum matchings would be compromised.

Future work will try to add more features to the layer tree without degrading its efficiency. It may be possible to incorporate information about workload limits into the resource sufficiency matchings, for example. A second goal is to design new timetabling algorithms that fully exploit the flexibility of this innovative data structure.

## References

1. Adriaen, M., De Causmaecker, P., Demeester, P., Vanden Berghe, G.: Tackling the university course timetabling problem with an aggregation approach. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, pp. 330–335 (August 2006)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, NJ (1993)
3. Apt, K.R.: *Principles of Constraint Programming*. Cambridge University Press, Cambridge (2003)
4. Burke, E.K., Carter, M. (eds.): *PATAT 1997*. LNCS, vol. 1408. Springer, Heidelberg (1998)

5. Burke, E.K., De Causmaecker, P. (eds.): PATAT 2002. LNCS, vol. 2740. Springer, Heidelberg (2003)
6. Burke, E., Erben, W. (eds.): PATAT 2000. LNCS, vol. 2079. Springer, Heidelberg (2001)
7. Burke, E.K., Ross, P. (eds.): Practice and Theory of Automated Timetabling. LNCS, vol. 1153. Springer, Heidelberg (1996)
8. Burke, E.K., Trick, M.A. (eds.): PATAT 2004. LNCS, vol. 3616. Springer, Heidelberg (2005)
9. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
10. Cooper, T.B., Kingston, J.H.: The solution of real instances of the timetabling problem. *The Computer Journal* 36, 645–653 (1993)
11. Fizzano, P., Swanson, S.: Scheduling classes on a college campus. *Computational Optimization and Applications* 16, 279–294 (2000)
12. van Hoeve, W.J.: A hyper-arc consistency algorithm for the soft alldifferent constraint. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 679–689. Springer, Heidelberg (2004)
13. Kingston, J.H.: A tiling algorithm for high school timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 233–249. Springer, Heidelberg (2005)
14. Kingston, J.H.: The KTS high school timetabling web site (Version 1.3) (October 2005), <http://www.it.usyd.edu.au/~jeff>
15. Kingston, J.H.: The KTS high school timetabling system. In: Burke, E.K., Rudová, H. (eds.) PATAT 2006. LNCS, vol. 3867, pp. 308–323. Springer, Heidelberg (2007)
16. Marte, M.: Towards constraint-based school timetabling. In: Proceedings of the Workshop on Modelling and Solving Problems with Constraints (at ICAI 2004), pp. 140–154 (2004)
17. Van Hentenryck, P.: *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA (1999)
18. de Werra, D.: An introduction to timetabling. *European Journal of Operational Research* 19, 151–162 (1985)

# The KTS High School Timetabling System

Jeffrey H. Kingston

School of Information Technologies,  
The University of Sydney, NSW 2006, Australia  
jeff@it.usyd.edu.au  
<http://www.it.usyd.edu.au/~jeff>

**Abstract.** KTS is a web-based high school timetabling system, freely accessible on the Internet. This paper is a general survey of KTS, including its data model, user interface, and solver. The solver uses operations research models in a polynomial-time heuristic framework to produce high quality solutions in a few seconds. Results are presented for six instances taken from Australian high schools.

## 1 Introduction

The problem of automatically constructing high school timetables has drawn the attention of researchers for many years. In the early days, *class-teacher timetabling* was the main focus [14], but it was never a realistic model of real-world problems. Progress on these has been slow. About ten years ago, a standard survey could find only a handful of papers reporting solvers for similar problems (including university course timetabling) in use in institutions [5]; and although firm data are hard to come by, the situation does not seem to have changed much since then.

Indeed, since that time the field seems to have been in decline; for example, the PATAT conferences of this period [12,3] contain only a few high school timetabling papers, and there are still no standard data sets on which researchers can compare results. Several recent papers, however, point to a renaissance; they attempt to solve real-world instances with such diverse methods as constraint programming [12], tabu search [8], and a hybrid approach [7]. These papers all relate to European high schools, which seem quite similar to the Australian high schools of this paper, the main differences being that European teachers are more likely to be preassigned to their classes, and are utilized less (around 50% of the time) than Australian teachers (around 75%), giving rise to compactness requirements not present in Australian problems.

This paper and the work it reports are motivated by a belief that, even if the technical problems can be overcome, software solvers will be widely used in high schools only if they are inexpensive and able to respond quickly to changing requirements. Such solvers would need to be available on-site under the control of the school's timetable planner, rather than off-site under the control of an expert (and hence expensive and busy) consultant.

KTS is a web server for high school timetabling created by the author. Its web interface puts the system on the desk of the school's timetable planner, and its polynomial time heuristic solver delivers a very good timetable in a few seconds. Together these features support non-traditional requirements such as rapid evaluation of alternative scenarios and incorporation of late changes, as well as the traditional one of solving a fixed instance to near-optimality. The system is fully operational and available continuously on the Internet [10], but at the time of writing the process of gathering a user community has only just begun (60 accounts have been created, but only a few are active).

This paper is a general overview of the KTS system. Section 2 presents a detailed specification of the high school timetabling problem as defined by KTS. Section 3 describes the user interface. Section 4 describes the solver, and Section 5 presents results for six instances taken from Australian high schools.

## 2 Data Model

The KTS data model is object-oriented. It is described in this section, with a few minor omissions.

An *account object*, or just *account*, represents one user's account with the KTS system. Each account contains any number of *institutions*, representing educational institutions for which the user wishes to construct timetables. Each institution contains any number of *instances*, each representing that institution's timetabling problem for a particular year, or semester, etc.

Each instance contains a *time group* object, holding all information about time. KTS has a simple time model in which time is divided into individual *times* of equal duration, ordered chronologically, with each time optionally separated from the next by a break, which could be a meal break or the end of a day, etc. The full sequence of times is called the *cycle*.

A sequence of one or more times that follow each other chronologically and do not span a break is called a *time block*. Any set of times may be viewed as a set of time blocks, by grouping the times into blocks of maximal size. The sizes of these blocks, written as a sequence of integers, form the *block structure* of the set of times. For example, the set of times  $\{Mon1, Mon2, Tue5, Tue6, Thu3\}$  presumably has block structure 2 2 1. The order in which the elements of a block structure are written does not matter; non-increasing order is used by convention. Meetings may specify that their times should have a particular block structure.

In addition to the instance's set of available times, the time group contains any number of *time subgroups*, which are subsets of the times, used when defining workload limits and *time conditions*. These latter place requirements on the sets of times assigned to meetings, and are either *limit conditions*, which limit the number of times from a given subgroup that a meeting may contain, for example limiting to 1 the number of undesirable times, or *spread conditions*, which require the time blocks assigned to a meeting to be spread evenly over a sequence of time subgroups, such as the days of the week.

An instance also contains any number of *resource group* objects, representing collections of *resources* (participants in meetings). Although not mandatory, there would typically be three resource groups, called *Student Groups*, *Teachers*, and *Rooms*. KTS is intended for high school timetabling problems, in which groups of students are timetabled, not individual students.

A resource group may contain *divisions*, representing administrative units such as faculties or departments (for teachers) and forms or years (for students). If a resource group has divisions, then each of its resources lies in exactly one of those divisions.

A resource group may also have *capabilities*, which are subsets of its set of resources. For example, an *English* capability would be the subset of teachers qualified to teach English; a *ScienceLab* capability would be the subset of rooms in which Science classes may be held. A resource may lie in any number of capabilities, and a capability may contain any number of resources. A division is usable as a capability, as is the resource group as a whole.

Each resource may have a set of times when it is unavailable to attend classes. It may also have *workload limits*, which might specify, for example, that the resource may attend meetings for at most 30 times over the cycle, and at most 7 times on each day. A limit may be placed on the number of occupied times in any subset of the times of the cycle, defined by a time subgroup. Each limit may have a *hard component*, a number of times which must not be exceeded, and a *soft component*, for which violations are penalized but not prohibited.

One resource may *follow* another. For example, a room may follow a particular teacher, meaning that it is to be preferred when assigning room selections in meetings to which that teacher is assigned. Such a room is often called the *home room* of a teacher.

An instance also contains *meetings*, which specify that certain resources are to meet together at certain times.

A meeting's times are specified by a single *time selection*, which requests that a particular number of times be assigned. It may request that the times conform to a given block structure, and include preassigned times. All time conditions defined in the time group apply to all time selections, as far as the time selection's block structure and preassigned times allow.

A meeting's resources are specified by any number of *resource selections*. For example, a meeting in which class 7A studies Science might contain a *Student Groups* resource selection requesting student group 7A, a *Teachers* resource selection requesting one teacher with the *Science* capability, and a *Rooms* selection requesting one *ScienceLab*. These selections may include preassigned resources.

An instance may contain any number of *solve profiles*, which are named collections of options for controlling the solver. The solver may be invoked with this set of options by a single click on the appropriate link. An instance may also contain any number of *display profiles*, which are named collections of options describing a timetable display or print: whether to use HTML, PDF, or PostScript;



**Meeting 7A-Science** [[Copy whole meeting](#)] [[Delete whole meeting](#)] [[New submeeting](#)]

**Times** Required blocks Suggested blocks Preassigned times  
 5 2 1 None (not selected) (not selected)

**Students** (preassigned only)  
 07A (not selected)

**Teachers** Capability Preassigned Special workload Splittable  
 1 ScienceYr7-10 (not selected) (not selected) Yes

**Rooms** Capability Preassigned Splittable  
 1 ScienceLab (not selected) Yes

New  Rooms,  Students,  Teachers resource select

7A-Science

**Fig. 1.** Screen shot of the user interface to one small meeting. A page header and navigation links precede this box and are not shown here. After the header line, the first inner box holds the time selection, here requesting 5 times including block structure 2 1. The next box holds a Student Groups resource selection, requesting student group resource 07A. This box accepts preassignments only, in accordance with an option set on the Student Groups resource group page. The following boxes request one ScienceYr7–10 teacher and one ScienceLab room. Split assignments are usually allowed; the Splittable boxes let the user disallow them for individual resource selections. Teachers have workload limits, so the Teachers selection offers a Special Workload box which allows the workload associated with this selection to be reduced (e.g., to 0 for staff meetings).

Bankstown Girls High School, 1998	Satisfactory		Unsatisfactory		Total	
	No.	%	No.	%	No.	%
<b>Assignments to time selections</b>						
Required blocks	<a href="#">147</a>	96.7	<a href="#">5</a>	3.3	<a href="#">152</a>	100.0
Suggested blocks	<a href="#">152</a>	100.0	<a href="#">0</a>	0.0	<a href="#">152</a>	100.0
At most 1 Undesirable time	<a href="#">152</a>	100.0	<a href="#">0</a>	0.0	<a href="#">152</a>	100.0
Even spread through Day 1 .. Day 5	<a href="#">120</a>	78.9	<a href="#">32</a>	21.1	<a href="#">152</a>	100.0
<b>Assignments to resource selections</b>						
Rooms	<a href="#">229</a>	97.9	<a href="#">5</a>	2.1	<a href="#">234</a>	100.0
Students	<a href="#">316</a>	100.0	<a href="#">0</a>	0.0	<a href="#">316</a>	100.0
Teachers	<a href="#">408</a>	92.3	<a href="#">34</a>	7.7	<a href="#">442</a>	100.0
<b>Soft workload limits</b>						
Teachers	<a href="#">266</a>	95.0	<a href="#">14</a>	5.0	<a href="#">280</a>	100.0

**Fig. 2.** Screen shot of the summary table from the evaluation page. Each underlined number is a link leading to a detailed list of defects. Below this table are other tables giving an intermediate level of detail, such as the number of time conditions defects affecting each student form, the number of soft workload overloads per teacher, etc.

whether to display large planning timetables or individual resources' timetables; whether to display the whole timetable, or just one division or resource; and so on. Again, one click produces a display using these options.

An instance may also contain a current solution. This consists of assignments of particular times and resources to some (hopefully all) of its time and resource selections. A resource assignment may be a *split assignment*, in which one qualified resource is assigned for some of the times of the meeting and a different one to the remaining times; or it may be a *partial assignment*, in which a particular resource is assigned for some of the times of a meeting but there is no assignment for the remaining times.

KTS objects are persistent: they exist permanently on disk, but can be updated in memory while the system is running. They are stored externally in UTF-8 text files, updated by a two-phase algorithm which protects against accidental corruption. Each account and its institutions occupies one file, and each instance occupies one file, including all the instance's objects (typically 10 to 20 kilobytes of data). Most operations concern a single instance, and they begin by reading this file and end by writing it. Instances are represented using a simple specification language, also called KTS, which is a descendant of the well-known TTL language [6]. The user may upload and download KTS instance files, although there is no strong motive for doing so.

### 3 User Interface

The KTS system is not distributed to users for installation on their own systems. Instead, there is a unique copy running at the author's institution, publicly accessible via the web, using HTML and CGI for its user interface. This has several advantages: it makes KTS available instantly on any computer connected to the Internet; the software may be upgraded centrally; and the data is held on the server where it may be captured for research purposes, in accordance with an agreement that users enter into when they create their accounts.

The user interface has one page for each object, beginning with a header and some navigation links, and continuing with updatable displays of the object's attributes. Most pages contain paragraphs of text describing their fields, so are self-documenting. The exception is the page which displays a meeting (Figure 1), where there is too much detail to document on the spot. Instead, a set of examples of meetings of increasing complexity is offered, which shows step-by-step how each meeting is built up. There is also an overview document explaining the capabilities of the system, and a glossary.

When there is a solution, KTS offers an evaluation page summarizing its defects (Figure 2), with links to more detailed evaluations. The most interesting of these detects sets of resource slots that cannot all be assigned to, owing to a shortage of resources (Figure 3).

Entry of a complete instance takes some hours. Short-cut operations for creating a time group and the usual resource groups help somewhat, as do operations for copying resources and meetings. There is also an operation for copying a complete instance, which saves time when moving to a new year or semester.

<b>Hall Set 2</b>				
The resource selections in this Hall set are:				
Meeting	Submeeting	Capability or resource	Required time(s)	Tixels
7CKO-D&T12-Art34	7CKO3-2	VisualArtsYr7-10	Mon5	1
7CKO-D&T12-Art34	7CKO4-2	VisualArtsYr7-10	Mon5	1
11-2/12-1	12-1-VisualArts	VisualArtsYr11-12	Mon5	1
<b>Total tixels demanded</b>				<b>3</b>
To satisfy this demand, the following tixels of supply are available:				
Resource	Time(s)	Tixels		
Diamond	Mon5	1		
Leon	Mon5	1		
<b>Total tixels supplied</b>		<b>2</b>		
Subtracting supply from demand gives 1 unassignable tixel in this Hall set.				

**Fig. 3.** Screen shot of a detailed evaluation, showing that a set of three simultaneous Art classes cannot all be assigned teachers, because there are only two Art teachers. The analysis is based on finding the Hall sets of a bipartite matching between all the tixels demanded by the instance and all the tixels supplied (a *tixel* is one resource at one time). Two versions of this analysis are carried out, one before time assignment and one after. Hall sets can be much more complex than this very simple example; they might reveal that the supply of English and History teachers, taken together, is insufficient to cover all the English and History classes even before time assignment, and so on. KTS merely prints the Hall sets; the user must find the explanations.

## 4 The Solver

The KTS solver aims to produce a very good and comprehensible timetable in ten seconds or less. It has five stages: *column layout*, *tile construction*, *time assignment*, *time adjustment*, and *resource assignment*. This approach appeared in an earlier paper by the author [9], but the algorithms presented there had many defects, as that paper acknowledged. With one exception (teacher assignment), the algorithms presented here are completely new; the defects are all removed, and more and better results are reported.

The following five subsections describe the five stages. Some details have been omitted, since a full description would be too lengthy for this paper, which aims to present a balanced view of the whole system.

### 4.1 Column Layout

As far as possible, the meetings in a high school timetable should overlap exactly in time, or not at all. This makes the timetable comprehensible, and simplifies resource assignment.

KTS's method of achieving such regularity begins by dividing the cycle into *columns*: sets of times which make good choices for assigning to meetings, and which meetings are encouraged to use wherever possible. The reader may be

	Day 1	Day 2	Day 3	Day 4	Day 5
Time 1	Column 1	Column 6	Column 2	Column 2	Column 5
Time 2	Column 1	Column 6	Column 2	Column 2	Column 5
Time 3	Column 6	Column 3	Column 4	Column 3	Column 3
Time 4	Column 6	Column 3	Column 4	Column 3	Column 6
Time 5	Column 5	Column 2	Column 1	Column 4	Column 4
Time 6	Column 5	Column 5	Column 1	Column 4	Column 7
Time 7	Column 4	Column 1	Column 5	Column 6	Column 7
Time 8	Column 2	Column 7	Column 3	Column 1	Column 7

**Fig. 4.** A typical layout of a week of 40 times into six columns of width 6 plus one of width 4. Breaks are not shown, but occur after the fourth and sixth times each day except Friday, when they occur after the third and fifth. This diagram was generated in PostScript by KTS.

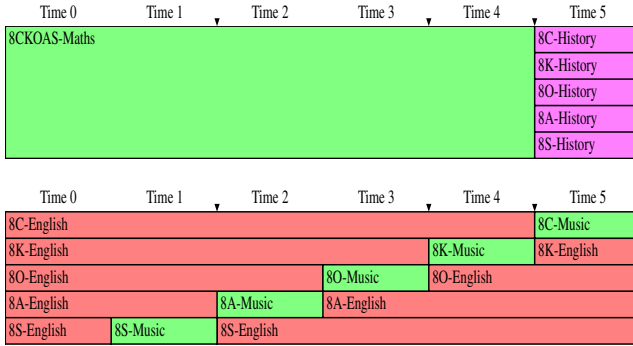
familiar with this approach from its use in North American universities, where the columns Mon–Wed–Fri 9–10AM, Mon–Wed–Fri 10–11AM, and so on, are frequently used. A traditional column plan in Australian high schools divides a cycle of 40 times into six columns each with six times, and one column with four times preassigned those times when the whole school attends Sport and optional religious instruction.

There is no requirement that meetings fit exactly into columns. In the senior years they usually do, but in the junior years the school offers many small subjects, often with little resemblance to any column plan.

Although a column plan could easily be inferred from the time selections of the meetings, it is such a basic part of the timetable planner’s thinking that it seems better to have the user enter it, including a number of times, block structure, and optional preassigned times for each column. Given this plan, the solver’s first task is to assign specific times to each column, aiming to ensure that each column satisfies the time conditions, so that meetings assigned to them will do so. An example of such a *column layout* appears in Figure 4. Producing it is quite easy in practice. The solver does it in two steps.

First, the time blocks naturally present in the cycle (between one break and the next) are partitioned into smaller blocks whose sizes exactly match the complete set of block sizes of the columns. KTS does this heuristically, checking after each break that the columns’ block sizes can be packed into the current cycle breakdown, and with an eye to the time conditions defined by the user: if meetings should be spread evenly over five days, then the solver aims to have the same number of time blocks on each day, and so on. Blocks of preassigned times already present in meetings are used wherever possible.

Second, the time blocks created by breaking down the cycle’s blocks are assigned to columns. After an initial round-robin assignment, a simple hill climber swaps pairs of equal-width time blocks between columns until no swap exists that reduces the badness of the columns as measured against the time conditions.



**Fig. 5.** Two examples of tiles from the *bghs98* instance. Each row is the timetable of one student group resource; each column is one time. The wedges indicate block structure.

### 4.2 Tile Construction

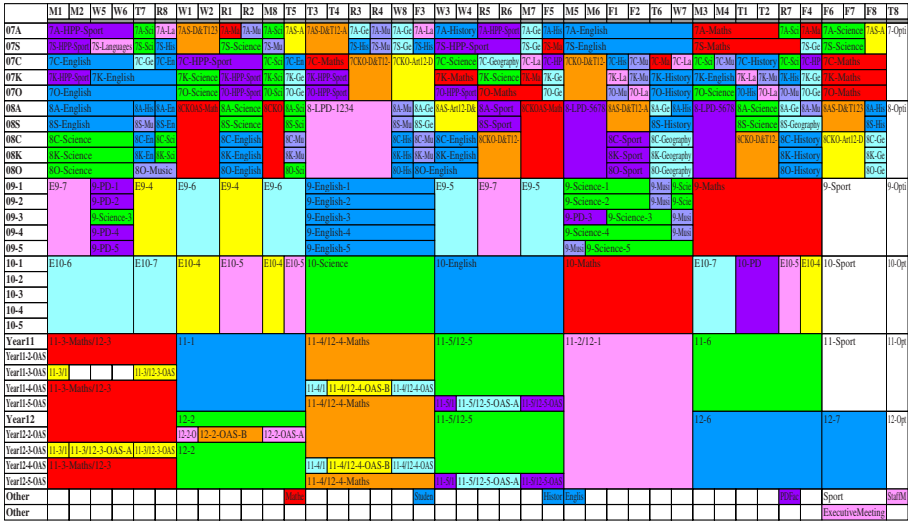
KTS continues its efforts to build a regular timetable by first timetabling small sets of meetings together into larger entities called *tiles*.

Figure 5 contains two examples of tiles. The students are grouped by ability for Mathematics, so the five Mathematics classes must run simultaneously and are combined into one large meeting in the input data. The adjacent History meetings do not have to run simultaneously, but fitting them neatly alongside Mathematics forces them to. The second tile illustrates a construction, well known to manual timetablers, called the *runaround*. There are only two Music teachers and two Music rooms, so the five Music classes cannot run simultaneously. By interleaving them among other meetings as shown, the tile demands only one of each at any one time.

Tiles are built in three steps. First, the meetings of each student form are grouped into *buckets*. Any meeting containing all the form’s student group resources goes into a bucket by itself; meetings which are identical except for their student group resources share a bucket; any meetings which cannot be analysed in a similar manner go into a leftovers bucket.

Second, a series of decisions is taken to merge certain sets of buckets. These decisions are made by a sequential heuristic which produces one merged bucket per iteration. Buckets that cannot be timetabled effectively because of a lack of resources are merged with other buckets. For example, the bucket holding the Music classes from Figure 5 is not viable alone and must be merged. Other relevant factors include preassigned times, the presence of student group resources from several forms, and a preference for tiles whose width (number of times) is a multiple of the usual column width, for regularity.

Finally, the meetings within each bucket are timetabled with respect to each other, producing tiles. This is a general time assignment problem, on a small scale, and the time assignment algorithm described in the next subsection is used to solve it. This step is interleaved with the previous one: if the bucket’s



**Fig. 6.** A planning timetable for the *bghs98* instance. Each row except the last two represents the timetable of one student group resource. The columns represent times, permuted to bring the times of the columns (in the column layout sense: six of width 6 and one of width 4) together, making them and the tiles within them clearly visible. An example of a time adjustment, swapping Science with Personal Development, appears in the row of student group 09-3. This diagram was generated in PostScript by KTS.

timetable turns out to be more defective than its meetings individually, the bucket merging heuristic tries alternative bucket mergings.

**4.3 Time Assignment**

After tiles are built, the next stage is to timetable them into the times of the cycle, producing a complete time assignment for all meetings.

The time assignment software module is called from three places within the KTS solver: to timetable submeetings into their meetings, meetings into their tiles, and tiles into the cycle. These problems are all essentially the same, differing only in scale. This description will speak of timetabling meetings into the cycle, rather than introducing unilluminating general terminology.

The meetings to be timetabled are first grouped into *layers*: sets of meetings required to be disjoint in time, typically because they contain the same pre-assigned student resources. The layers are sorted so that the most difficult ones (those requiring the most resources) come first, and timetabled one by one with no backtracking. A meeting may lie in more than one layer, in which case it is timetabled along with its first layer. In the time assignment stage of the solver there is one layer per student form, plus one layer for each staff meeting.

Within each layer, each meeting is timetabled in turn, widest first, if possible into a single column. A few assignments are tried for each meeting, but without backtracking; instead, forward checks, involving two kinds of bipartite matchings that monitor the availability of resources, keep the solver on track. These checks are described in detail in a companion paper [11]. A timetable created by this algorithm, plus time adjustment, appears in Figure 6.

#### 4.4 Time Adjustment

After a complete time assignment is obtained, *time adjustment* attempts to improve it by hill climbing: swapping time blocks around while this produces an improvement. Hill climbing is very effective here, since it corrects simple problems resulting from the lack of backtracking during time assignment, in time proportional to the number of improvements it makes (multiplied by the neighbourhood size).

Although no resources have yet been assigned to meetings, there are nevertheless two useful evaluations that can be made at this point: checking the sets of times assigned to meetings for their conformance to time conditions, and checking that resources are sufficient at each time to cover the resource demands made by meetings assigned that time (using a bipartite matching at each time between resource demands and resources). A neighbour is accepted if it reduces problems with resources, or improves time conditions without increasing problems with resources.

There are several promising neighbourhoods that could be tried. The current implementation explores two, repeating until neither gives any improvement. Both swap the members of a set of equal-width non-overlapping time blocks, so the neighbourhood size is bounded by the square of the number of times in the cycle, and is often much less.

The first neighbourhood takes each pair of time blocks of equal size assigned to columns, such that none of the times involved is preassigned to any meeting or column, and tries swapping these time blocks globally through every meeting. This might reduce resource problems as well as time condition problems, because resources' unavailable times stay fixed, and a swap might move resource demands away from the unavailable times of the resources they need.

The second neighbourhood takes pairs of meetings that contain the same preassigned resources (typically student group resources) and swaps blocks of their times of equal width. Since this can disrupt the regularity of a timetable, these swaps are only accepted if they reduce problems with resources, and indeed are only tried at times where there are such problems.

#### 4.5 Resource Assignment

*Resource assignment* is the assignment of particular resources to the resource slots of meetings. The solver does this after times are all assigned.

Each resource group may be assigned independently of the others, apart from a slight connection caused by 'follows' requirements. For each resource group in

	Avail	M1	M2	W5	W6	T7	R8	W1	W2	R1	R2	M8	T5	T3	T4	R3	R4	W8	F3	W3	W4	R5	R6	M7	F5	
Gibbons	0	8C-Science		7A-Sci		7A-Sci	8A-Science	7A-Science		8A-Science	7A-Sci	8A-Sci							Student	12-5-Chemistry						
Kassab	0							7O-Science		8S-Science	7O-Sci	8S-Sci	10-Science2						11-5-Biology							
Kidd	0	12-3-Physics										7C-Sci	10-Science1						7C-Science							
Prasad	0	8K-Science						12-2-GeneralScience-1				10-Science3														
Saule	0	8O-Science						12-2-Biology				10-Science5								12-5-Biology						
Smith	1			9-Science-3		7S-Sci			7S-Science		8O-Sci	10-Science4								11-5-Physics						
Unassigned												7A-Sci													7K-Science	

	M5	M6	F1	F2	T6	W7	M3	M4	T1	T2	R7	F4	F6	F7	F8	T8
Gibbons	9-Science-1						9-Scienc			8A-Science		7A-Sci	Sport		StaffMe	
Kassab	11-2-GeneralScience						7O-Science		8S-Science							
Kidd	9-Science-5						7C-Sci				7C-Sci	ExecutiveMeeting				
Prasad	9-Science-4						11-6-Chemistry-1				Sport					
Saule							11-6-Chemistry-2									
Smith			9-Science-3				11-6-Biology				7S-Science					
Unassigned																

**Fig. 7.** Planning timetable showing the teacher assignment for the Science faculty of the *bghs98* instance. (The resource assignment algorithm assigns all faculties simultaneously, but it is convenient to analyse its results faculty by faculty.) The second column gives the remaining unused workload of each teacher. Split and partial assignments are shown in italic font. There are three unassigned tixels. This diagram was generated in PostScript by KTS.

turn, in an order influenced by the presence of ‘follows’ requirements, preassignments are first converted to assignments, then assignments arising from ‘follows’ requirements are made, then all remaining unassigned slots are assigned. Some preassignments may fail to convert owing to resource unavailabilities and workload limits; their slots remain unassigned and become defects in the solution.

The resource assignment problem comes in two versions, depending on how acceptable split assignments are. Typically, split assignments are undesirable when assigning teachers, but acceptable when assigning rooms, provided classes do not have to change rooms part-way through a time block.

The room assignment algorithm is quite simple. It assigns each time block of each meeting sequentially, largest blocks first, choosing a qualified resource whose use does not increase the number of resource problems at any of the block’s times (the usual bipartite matching checks this condition), and preferring a resource which has already been assigned to another block of the meeting. If a block of two or more times is encountered for which this is not possible, it is split into blocks of width 1; if a block of width 1 cannot be assigned, it is passed over and becomes a defect in the solution.

The teacher assignment algorithm tries much harder to avoid split assignments (Figure 7). It is based on the alternating path method familiar from bipartite matching and similar problems, used as a heuristic, since the optimality guarantees that usually accompany it are absent.



**Table 1.** The six instances tested, showing the number of times, resources, and meetings in each

Instance	Times	Student groups	Teachers	Rooms	Meetings
<i>bghs93</i>	40	23	53	46	155
<i>bghs95</i>	40	27	52	48	147
<i>bghs98</i>	40	30	56	45	152
<i>tes98</i>	30	11	33	20	95
<i>tes99</i>	30	13	37	26	86
<i>sahs96</i>	60	20	43	36	131

Choose a currently unassigned teacher slot of maximum width. If there is a qualified teacher able to fill this slot (i.e. without causing clashes or exceeding workload limits), assign that teacher and move to the next widest slot. Otherwise, see if there is a teacher who could fill the slot if only some one of the assignments currently given to that teacher were deassigned and given to some other teacher able to fill it. If so, make the indicated chain of two assignments and one deassignment, and move on. If not, look for a longer chain, and so on. At each moment when there are no workload overloads or clashes, compare the whole set of assignments with the best so far, and replace it if it is better.

Two methods of controlling the size of the search are used. One is the traditional one of marking each possible assignment and deassignment *visited* when it is first considered, and refusing to reconsider it during the course of the search (it becomes available again when we move to the next slot). The other method is to allow revisiting but to strictly limit the depth of the search, to the empirically determined value of 5 (three assignments and two deassignments). The searches are repeated until there is no improvement.

At each slot, in addition to searching for ordinary assignments, the solver finds a qualified resource which is available for as many times as possible, and generates all split assignments which have that resource and those times as the first branch, and one other qualified resource with the remaining times as the second branch. The alternating path search continues down the second branch. A single partial assignment is also generated, holding the first branch as before but omitting the second.

## 5 Results

At the time of writing, KTS has only recently been released to the Internet, and no freshly created instances are available. In default of those, this section analyses the performance of the solver on six real instances from the author's archives, taken from three high schools in Sydney, Australia. These have been solved without any of the interactive exploration of alternative scenarios that KTS makes possible. The results are offered as evidence that the approach taken by the solver is promising; the small number of instances and the unavailability

**Table 2.** Run times in seconds for the major stages and in total. The tests used a 3.2GHz Pentium machine running Linux. Run times are as reported by the Linux *time* command, which is accurate to one second. Column layout time was always 0 seconds so has been omitted. Time assignment includes time adjustment by hill climbing, never more than one second. The times given for resource assignment essentially measure teacher assignment only, since room assignment is very fast. Total times were checked against wristwatch time.

Instance	Tile construction	Time assignment	Resource assignment	Total
<i>bghs93</i>	0	3	3	6
<i>bghs95</i>	0	1	7	8
<i>bghs98</i>	0	1	6	7
<i>tes98</i>	1	1	0	2
<i>tes99</i>	0	1	0	1
<i>sahs96</i>	1	31	0	32

**Table 3.** Evaluation of time assignments, showing the absolute number of meetings with defective block structure, uneven spread through the cycle, and more than one undesirable time, plus this number as a percentage of the total number of meetings

Instance	Block structure	Spread	Undesirable times
<i>bghs93</i>	48 (31.0%)	51 (31.2%)	-
<i>bghs95</i>	20 (13.6%)	44 (29.9%)	7 (4.8%)
<i>bghs98</i>	5 (3.3%)	31 (21.1%)	0 (0.0%)
<i>tes98</i>	36 (37.9%)	22 (23.2%)	2 (2.1%)
<i>tes99</i>	37 (43.0%)	27 (31.4%)	-
<i>sahs96</i>	2 (1.5%)	74 (56.5%)	18 (13.7%)

of comparisons with other solutions (especially hand-generated ones) rule out larger claims.

Statistical descriptions of the six instances appear in Table 1, run times are given in Table 2, and the quality of the solutions is summarized in Tables 3, 4, and 5. The solver always assigns the correct number of times to each meeting, never introduces student group clashes, and prefers to leave teacher and room slots unassigned rather than introducing teacher and room clashes and workload overloads. So the possible defects are time assignment problems (wrong block structure, meeting spread over too few days, etc.) and unsatisfactory room and teacher assignments (split, partial, and missing).

The *sahs96* instance has a two-week cycle, and all its teacher slots are pre-assigned. These two factors make time assignment very slow. It is encouraging that only 3.1% of these preassigned teacher tixels could not be assigned (Table 5), given that the solver is not optimized to handle instances that are highly constrained in this way. However, the solver's desperate attempt to satisfy all these preassignments leads to a quite irregular timetable.

**Table 4.** Evaluation of room assignments, showing the absolute number of split assignments, partial and missing assignments, unassignable tixels after time assignment (1), and unassigned tixels after resource assignment (2), plus this number as a percentage of the number of room assignments or tixels demanded. In this table, a split assignment is one in which a class has to change rooms part-way through a time block.

Instance	Split	Partial/missing	Tixels (1)	Tixels (2)
<i>bghs93</i>	0 (0.0%)	7 (3.1%)	15 (1.2%)	15 (1.2%)
<i>bghs95</i>	0 (0.0%)	6 (2.9%)	9 (0.7%)	9 (0.7%)
<i>bghs98</i>	0 (0.0%)	5 (2.1%)	7 (0.5%)	7 (0.5%)
<i>tes98</i>	2 (2.2%)	5 (5.5%)	7 (1.5%)	7 (1.5%)
<i>tes99</i>	0 (0.0%)	5 (3.7%)	7 (1.3%)	7 (1.3%)
<i>sahs96</i>	0 (0.0%)	27 (11.2%)	15 (1.0%)	15 (1.0%)

**Table 5.** Evaluation of teacher assignments, showing the absolute number of split assignments, partial and missing assignments, unassignable tixels after time assignment (1), and unassigned tixels after resource assignment (2), plus this number as a percentage of the number of teacher assignments or tixels demanded, as appropriate. In this table, a split assignment is one in which a class is taught by two teachers.

Instance	Split	Partial/missing	Tixels (1)	Tixels (2)
<i>bghs93</i>	3 (0.7%)	7 (1.5%)	5 (0.3%)	9 (0.6%)
<i>bghs95</i>	17 (3.7%)	15 (3.3%)	7 (0.5%)	27 (2.0%)
<i>bghs98</i>	24 (5.4%)	10 (2.3%)	8 (0.5%)	17 (1.2%)
<i>tes98</i>	7 (3.8%)	13 (7.1%)	14 (3.0%)	14 (3.0%)
<i>tes99</i>	2 (1.1%)	9 (5.1%)	9 (1.7%)	9 (1.7%)
<i>sahs96</i>	0 (0.0%)	27 (11.2%)	47 (3.1%)	47 (3.1%)

The other instances are more typical of the solver's intended domain of application. Run times are under ten seconds. Block structure defects are somewhat high (Table 3). This problem awaits analysis but should be correctable. Time conditions defects are probably acceptable now, given their relative unimportance, although there is room for improvement.

Resource assignment can be evaluated either in terms of the number of defective assignments (split, partial, or missing), or the number of unassigned individual tixels (a *tixel* is one resource at one time, either supplied or demanded). Some tixels are inevitably unassignable given a particular time assignment – for example, if the time assignment requires five Science laboratories to be available at some time, but the school has only four. These are shown in the fourth column of Tables 4 and 5, while the number of unassigned tixels after resource assignment is shown in the fifth column.

Room assignment (Table 4) is virtually perfect. The room assignment algorithm always assigns every room tixel that time assignment permits, because it breaks time blocks up into individual times if necessary, and, using a bipartite

matching between room demands and rooms at each time, it never allows the number of unassignable rooms at any time to increase. This is why the fourth and fifth columns of Table 4 are equal. The fact that only two split assignments were ever introduced shows how easy this problem is in practice, despite being formally NP-complete [4].

Unassigned room tixels typically request specialized laboratories whose demand is very tight. This problem is quite common in high schools and is not of major concern, since, given its low relative frequency, it is not difficult to ensure that no class meets in an inappropriate room for more than one of its times, and the teacher would organize the classroom material accordingly. An option to assign inappropriate rooms where necessary, spreading them fairly among the classes affected, could easily be added.

Split teacher assignments and unassigned teacher tixels (Table 5) are the main areas of concern. How acceptable these results are it is hard to say. Hand-generated timetables also have these problems. Split assignments are quite routine. Unassigned tixels are handled in various ways: by excusing a teacher from a faculty meeting, having an available but unqualified teacher supervise a class, and so on. Unlike other defects, every unassigned teacher tixel is a real problem requiring the attention of the timetable planner.

One unassignable tixel in a teacher slot spoils the assignment of the entire slot. This suggests that finding time assignments with fewer unassignable teacher tixels would be more helpful than improving the teacher assignment algorithm.

## 6 Conclusions

This paper has presented KTS, a freely accessible web-based system for high school timetabling. It has demonstrated a new and better way to deliver fast, effective, and inexpensive high school timetabling.

The fast response time makes KTS well suited to exploring alternative scenarios and incorporating late changes to requirements. However, KTS does not yet address the problem of making minimal changes to a published solution in response to changes in requirements [13].

The data model is mature, except perhaps in its treatment of time, and the overall structure of the solver is quite successful. It seems likely that future work will focus on improving the existing solver components, rather than radically redesigning the solver. The time assignment stage is the obvious next target for improvement. In fact, since this paper was written, the author has designed and implemented a more flexible approach to time assignment and adjustment which should allow the algorithms described here to be varied and generalized in several interesting ways [11].

In parallel with these efforts, the KTS system will be promoted to Australian high schools. More users will bring a larger and more diverse set of test instances, which should lead to further progress.

## References

1. Burke, E., Erben, W. (eds.): PATAT 2000. LNCS, vol. 2079. Springer, Heidelberg (2001)
2. Burke, E.K., De Causmaecker, P. (eds.): PATAT 2002. LNCS, vol. 2740. Springer, Heidelberg (2003)
3. Burke, E.K., Trick, M.A. (eds.): PATAT 2004. LNCS, vol. 3616. Springer, Heidelberg (2005)
4. Carter, M.W., Tovey, C.A.: When is the classroom assignment problem hard? *Operations Research* 40, S28–S39 (1992)
5. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
6. Cooper, T.B., Kingston, J.H.: The solution of real instances of the timetabling problem. *The Computer Journal* 36, 645–653 (1993)
7. de Haan, P., Landman, R., Post, G., Ruizenaar, H.: A four-phase approach to a timetabling problem in secondary schools. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, pp. 423–425 (August 2006)
8. Jacobsen, F., Bortfeldt, A., Gehring, H.: Timetabling at German secondary schools: tabu search versus constraint programming. In: Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, pp. 439–442 (August 2006)
9. Kingston, J.H.: A tiling algorithm for high school timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 208–225. Springer, Heidelberg (2005)
10. Kingston, J.H.: The KTS high school timetabling web site (Version 1.3) (October 2005), <http://www.it.usyd.edu.au/~jeff>
11. Kingston, J.H.: Hierarchical Timetable Construction. In: Burke, E.K., Rudová, H. (eds.) PATAT 2006. LNCS, vol. 3867, pp. 294–307. Springer, Heidelberg (2007)
12. Marte, M.: Towards constraint-based school timetabling. In: Proceedings of the Workshop on Modelling and Solving Problems with Constraints (at ECAI 2004), pp. 140–154 (2004)
13. Müller, T., Rudová, H., Barták, R.: Minimal perturbation problem in course timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 126–146. Springer, Heidelberg (2005)
14. Schmidt, G., Ströhlein, T.: Timetable construction – an annotated bibliography. *The Computer Journal* 23, 307–316 (1980)

# **Examination Timetabling**

# A Novel Fuzzy Approach to Evaluate the Quality of Examination Timetabling

Hishammuddin Asmuni<sup>1</sup>, Edmund K. Burke<sup>1</sup>, Jonathan M. Garibaldi<sup>1</sup>,  
and Barry McCollum<sup>2</sup>

<sup>1</sup> School of Computer Science and Information Technology,  
University of Nottingham, Jubilee Campus,  
Wollaton Road, Nottingham, NG8 1BB, UK  
{hba, ekb, jmg}@cs.nott.ac.uk

<sup>2</sup> School of Computer Science, Queen's University Belfast,  
Belfast BT7 1NN, UK  
b.mccollum@qub.ac.uk

**Abstract.** In this paper we introduce a new fuzzy evaluation function for examination timetabling. We describe how we employed fuzzy reasoning to evaluate the quality of a constructed timetable by considering two criteria: the average penalty per student and the highest penalty imposed on any of the students. A fuzzy system was created based on a series of easy to understand rules to combine the two criteria. A significant problem encountered was how to determine the lower and upper bounds of the decision criteria for any given problem instance, in order to allow the fuzzy system to be fixed and, hence, applicable to new problems without alteration. In this work, two different methods for determining boundary settings are proposed. Experimental results are presented and the implications analysed. These results demonstrate that fuzzy reasoning can be successfully applied to evaluate the quality of timetable solutions in which multiple decision criteria are involved.

## 1 Introduction

Timetabling refers to the process of allocating limited resources to a number of events subject to many constraints. Constraints are divided into two types: hard and soft. Hard constraints cannot be violated under any circumstances. Any timetable solution that satisfies all the specified hard constraints is considered to be a *feasible solution*, provided that all the events are assigned to a time slot. Soft constraints are highly desirable to satisfy, but it is acceptable to breach these types of constraint. However, it is very important to minimise the violation of the soft constraints, because, in many cases, the quality of the constructed timetable is evaluated by measuring the fulfillment of these constraints. In practice, the variety of constraints which are imposed by academic institutions are very different [6]. Such variations make the timetabling problem more challenging. Algorithms or approaches that have been successfully applied to one problem may not perform well when applied to different timetabling instances.

Researchers have employed many different approaches over the years in an attempt to generate ‘optimal’ timetabling solutions subject to a list of constraints. Approaches such as Evolutionary Algorithms, e.g. [8, 16, 28], Tabu Search, e.g. [7, 17, 19, 29], Simulated Annealing, e.g. [27], Constraint Programming, e.g. [1, 4, 18], Case-Based Reasoning, e.g. [11, 30], and Fuzzy Methodologies, e.g. [2, 3, 23, 30] have been successfully applied to timetabling problems. Overviews of timetabling approaches are presented in [10, 12, 22, 24, 26].

In 1996, Carter et al. [13] introduced a set of examination timetabling benchmark data. The original benchmark data set consists of 13 problem instances. Since then certain difficulties have come to light with these benchmarks because different versions circulated under the same name (the situation is discussed and clarified in [24]. However, these benchmarks remain an important testbed. They consider the following constraints:

**Hard constraint.** The constructed timetable must be conflict free. The requirement is to avoid any student being scheduled for two different exams at the same time.

**Soft constraint.** The solution should attempt to minimise the number of exams assigned in adjacent time slots in such a way as to reduce the number of students sitting exams in close proximity.

In the context of these benchmark data sets, several different objective functions have been introduced in order to measure the quality of the timetable solution. In addition to the commonly used objective function that evaluates only the proximity cost (see next section for details), other objective functions have been derived based on the satisfaction of other soft constraints, such as minimising consecutive exams in one day or overnight, assigning large exams to early time slots, and others. This is discussed in more detail in the following section.

Previous studies such as [3] and [23], demonstrated that fuzzy reasoning is a promising technique that can be used both for modelling timetabling problems and for constructing solutions. These studies indicated that the utilisation of fuzzy methodologies in university timetabling is an encouraging research topic. In this paper, we introduce a new evaluation function that is based on fuzzy methodologies. The research presented in this paper will focus on evaluating the constructed timetable solutions by considering two decision criteria. Although the constructed timetable solutions were developed based on objectives specified earlier, the method is general in the sense that a user could, in principle, define additional criteria to be taken into account in evaluating any constructed timetables. This paper is motivated by the fact that, in practice, the quality of the timetable solution is usually assessed by a timetabling officer who considers several criteria/objectives.

In the next section, we present a brief description of existing evaluation methods, their drawbacks, and a detailed explanation of the proposed novel approach. Section 3 presents descriptions of the experiments carried out and the results obtained, followed by discussions in Section 4. Finally, some concluding comments and future research directions are given in Section 5.



## 2 Assessing Timetable Quality

### 2.1 Existing Evaluation Functions

This section presents several evaluation functions that have been developed for Carter et al.'s benchmark data sets. The proximity cost function was the first evaluation function used to measure the quality of timetables [13]. It is motivated by the goal of spreading out each student's examination schedule. In the implementation of the proximity cost, it is assumed that the timetable solution satisfies the defined hard constraint i.e. no student can attend more than one exam at the same time. In addition, the solution must be developed in such a way that it will promote the spreading out of each student's exams so that students have as much time as possible between exams. If two exams scheduled for a particular student are  $t$  time slots apart, a penalty weight is set to  $w_t = 2^{5-t}$  where  $t \in \{1, 2, 3, 4, 5\}$  (as implemented in [13] and widely adopted by most subsequent research in this area). The weight is multiplied by the number of students that sit both the scheduled exams. The average penalty per student is calculated by dividing the total penalty by the total number of students. The maximum number of time slots for each data set is predefined and fixed, but no limitation in terms of capacity per time slot is set. Consecutive exams, either in the same day or overnight, are treated the same, and there is no consideration of weekends or other actual gaps between logically consecutive days. Hence, the following formulation is used to measure this proximity cost (see, for example, Burke et al. [5]):

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

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

Burke et al. [8] devised a new evaluation function in which the goal is to minimise the number of students who have to sit two exams in the same day. Besides the need to construct a conflict-free timetable, it also aimed to schedule the exams within the maximum number of time slots given. There are three time slots per weekday and one morning slot on Saturday. A maximum capacity per time slot is also specified. Burke and Newall [9] extended the previous evaluation function by defining different weights for two consecutive exams in the same day and two exams in overnight consecutive time slots.

More recently, Petrovic et al. [23] employed fuzzy methodologies to measure the satisfaction of various soft constraints. The authors described how they modeled two soft constraints, namely *constraint on large exam* and *constraint on proximity of exams*, in the form of fuzzy linguistic terms and defined the related rule set. They used these two criteria to evaluate the timetable quality.

## 2.2 Disadvantages/Drawbacks of Current Evaluation Functions

As can be seen, the final value of the proximity cost penalty function is a measure only of the average penalty per student. Although this penalty function has been widely used by many researchers in the context of the benchmark data set, in practice, considering only the average penalty per student is not sufficient to evaluate the quality of the constructed timetable. For instance, the final value does not necessarily represent the relative fairness of spreading out each student's schedule. For example, when examining the resultant timetable, it may be the case that a few students have an examination timetable in which many of their exams are scheduled in adjacent time slots. These students will not be happy with their timetable as they will not have enough time to do their preparation. On the other hand, the remaining students enjoy a 'good' examination timetable.

*Example.* Consider two cases. Case 1: there are 100 students with each student given 1 penalty cost; Case 2: there are 100 students, but now 10 students are given 10 penalty cost respectively; the rest zero. In both cases the average penalty per student is equal to 1, but obviously the solution in Case 2 is 'worse' than the solution in Case 1.

One of the authors (McCollum) has extensive experience of real-world timetabling, having spent 12 years as a timetabling officer and with continuing links with the timetabling industry. He has expressed (via private communication) the view that 'proximity cost' is not the only factor considered by timetabling officers when evaluating the quality of a timetable. Usually, a timetable evaluation is based on several factors and some of the factors are subjective and/or based on ambiguous information. Furthermore, to the best of our knowledge, all the evaluation functions mentioned in Section 2.1 are integrated into the timetabling construction process. These objective functions are used to measure the satisfaction of specific soft constraints. This means that the constructed timetable is generated around satisfying certain soft constraints. In practice, the user may consider other criteria in evaluating the constructed timetable.

One way to handle multiple criteria decision making is by using simple linear combinations. This works by multiplying the value of each criterion by a constant weighting factor and summing to form an overall result. Each weight represents the relative important of each criterion compared to the other criteria. In reality, there is no simple way to determine the precise values for these weights, especially weights that can be used across several problem instances with different complexity. Fuzzy systems are a generalisation of a linear system. The nature of fuzzy systems allows the use of linguistic terms to express the systems' behaviours. Fuzzy systems apply 'if-then' rules and logical operators to map the relationships between input and output variables in the system. Fuzzy rules may be elicited from 'experts'. This term, for the problem under consideration, refers to timetabling officers or timetabling consultants. As mentioned earlier, we have access to such experts who could provide us with enough knowledge to develop a fuzzy system.

Therefore, in this paper a new evaluation function utilising fuzzy methodologies is introduced. Basically, the idea is to develop an independent evaluation function that can be used to measure the quality of any constructed examination timetable. The timetable can be generated using any approach with specific objectives to achieve. Subsequently, the timetable solution with the problem description and the list of factors that need to be evaluated are submitted to the evaluation function.

### 2.3 Overview of Fuzzy Systems

This section is largely reproduced from our paper [3] for the purpose of completeness. In many decision making environments, it is often the case that several factors are simultaneously taken into account. Often, it is not known which factor(s) need to be emphasised more in order to generate a better decision. Somehow a trade-off between the various (potentially conflicting) factors must be made. The general framework of fuzzy reasoning facilitates the handling of such uncertainty.

Fuzzy systems are used for representing and employing knowledge that is imprecise, uncertain, or unreliable. A fuzzification component computes the membership grade for each crisp input variable based on the membership functions defined. An inference engine then conducts the fuzzy reasoning process by applying the appropriate fuzzy operators in order to obtain the fuzzy set to be accumulated in the output variable. A defuzzifier transforms the output fuzzy set to crisp output by applying specific defuzzification method.

More formally, a fuzzy set  $A$  of a universe of discourse  $X$  (the range over which the variable spans) is characterised by a *membership function*  $\mu_A : X \rightarrow [0, 1]$  which associates with each element  $x$  of  $X$  a number  $\mu_A(x)$  in the interval  $[0, 1]$ , with  $\mu_A(x)$  representing the *grade of membership* of  $x$  in  $A$  [31]. The precise meaning of the membership grade is not rigidly defined, but is supposed to capture the ‘compatibility’ of an element to the notion of the set. Rules which connect input variables to output variables in ‘IF ... THEN ...’ form are used to describe the desired system response in terms of *linguistic* variables (words) rather than mathematical formulae. The ‘IF’ part of the rule is referred to as the ‘antecedent’, the ‘THEN’ part is referred to as the ‘consequent’. The number of rules depends on the number of inputs and outputs, and the desired behaviour of the system. Once the rules have been established, such a system can be viewed as a non-linear mapping from inputs to outputs.

There are many alternative ways in which this general fuzzy methodology can be implemented in any given problem. In our implementation, the standard Mamdani style fuzzy inference was used with standard Zadeh (min-max) operators. In Mamdani inference [20], rules are of the following form:

$$R_i : \text{if } (x_1 \text{ is } A_{i1}) \text{ and } \dots \text{ and } (x_r \text{ is } A_{ir}) \text{ then } (y \text{ is } C_i) \text{ for } i = 1, 2, \dots, L$$

where  $L$  is the number of rules,  $x_j$  ( $j = 1, 2, 3, \dots, r$ ) are input variables,  $y$  is output variable, and  $A_{ij}$  and  $C_i$  are fuzzy sets that are characterised by

membership functions  $A_{ij}(x_j)$  and  $C_i(y)$ , respectively. The final output of a Mamdani system is one or more arbitrarily complex fuzzy sets which (usually) need to be defuzzified. It is not appropriate to present a full description of the functioning of fuzzy systems here; the interested reader is referred to [21] and [15] for a simple treatment or Zimmerman [32] for a more complete treatment.

## 2.4 The Proposed Fuzzy Evaluation Function

As an initial investigation, this proposed approach was implemented on solutions which were generated based on the proximity cost requirements (*average penalty*), with one additional factor/objective. In addition to the average penalty per student, the highest penalty that occurred amongst the students (*highest penalty*) was also taken into account. However, the latter factor was only evaluated after the timetable was constructed. That is to say, there was no attempt to include this factor in the process of constructing the timetable.

A fuzzy system with these two input variables (*average penalty* and *highest penalty*) and one output variable (*quality*) was constructed. Each of the input variables were associated with three linguistic terms: fuzzy sets corresponding to a meaning of *low*, *medium* and *high*. In addition to these three linguistic terms, the output variable (*quality*) has two extra terms that correspond to meanings of *very low* and *very high*. These terms were selected as they were deemed to be the simplest possible to adequately represent the problem. Gaussian functions of the form  $e^{-(x-c)^2/\sigma^2}$ , where  $c$  and  $\sigma$  are constants, are used to define the fuzzy set for each linguistic term. This is on the basis that they are the simplest and most common choice, given that smooth, continuously varying functions were desired. The membership functions defined for the two inputs, *average penalty* and *highest penalty*, and the output *quality* are depicted in Figures 1(a)–(c), respectively.

In the case of such a system having two inputs with three linguistic terms, there are nine possible fuzzy rules that can be defined in which each input variable has one linguistic term. As we already know, from the definition of proximity cost, the objective is to minimise the penalty cost, meaning that, the lower the penalty cost, the better the timetable quality. Also, based on everyday experience, we would prefer the highest penalty for any one student to be as low as possible, as this will create a fairer timetable for all students. Based upon this knowledge we defined a fuzzy rule set consisting of all nine possible combinations. Each rule set connects the input variables to a single output variable, *quality*. The fuzzy rule set is presented in Figure 2. As stated above, standard Mamdani-style fuzzy inference was used to obtain the fuzzy output for a given set of inputs. The most common form of defuzzification, ‘centre of gravity defuzzification’, was then used to obtain a single crisp (real) value for the output variable. This process is based upon the notion of finding the centroid of a planar figure, as given by

$$\sum_i \frac{\mu(x_i) \cdot x_i}{\mu(x_i)}.$$

This single crisp output was then taken as the *quality* of the timetable.

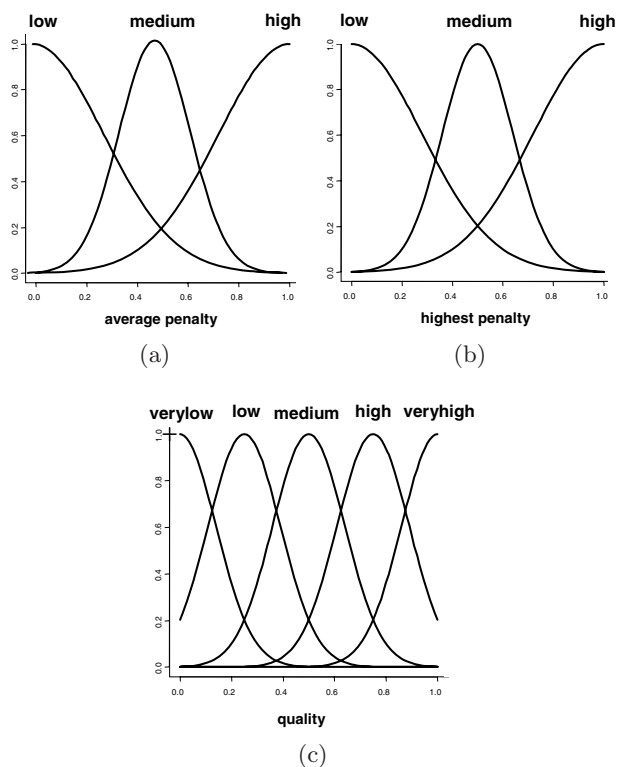


Fig. 1. Membership functions for input and output variables

## 2.5 Input Normalisation

With this proposed fuzzy evaluation function, we carried out experiments to determine whether the fuzzy evaluation system was able to distinguish a range of timetable solutions based on the average penalty per student and the highest penalty imposed on any of the students. All the constructed timetables for the given problem instance were evaluated using the same fuzzy system, and their quality determined based on the output of the fuzzy system. The constructed timetable with the biggest output value was selected to be the ‘best’ timetable.

Based on our previous experience [2,3], the average penalty values for different data sets result in widely different scales due to the different complexity of the problem instances. For example, in the STA-F-83I data set (see below for full details of the data sets used) an average penalty of 160.42 was obtained, whereas for UTA-S-92I, the average penalty was 3.57.

As can be seen in Figure 1(a) and Figure 1(b), the input variables have their universe of discourse defined between 0.0 and 1.0. Therefore, in order to use this fuzzy model, both of the original input variables must be normalised within the range  $[0.0, 1.0]$ . The transformation used is as follows:

$$v' = \frac{(v - \text{lowerBound})}{(\text{upperBound} - \text{lowerBound})}$$

where  $v$  is the actual value in the initial range  $[\text{lowerBound}, \text{upperBound}]$ . In effect, the range  $[\text{lowerBound}, \text{upperBound}]$  represents the actual lower and upper boundaries for the fuzzy linguistic terms.

By applying the normalisation technique, the same fuzzy model can be used for any problem instance, either for the benchmark data sets as used here, or for a new real-world problem. This would provide flexibility when problems of various complexity are presented to the fuzzy system. In such a scheme, the membership functions do not need to be changed from their initial shapes and positions. In addition, rather than recalculate the parameters for each input variable's membership functions, it is much easier to transform the crisp input values into normalised values in the range of  $[0.0, 1.0]$ . The problem thus becomes one of finding suitable lower and upper bounds for each problem instance.

### 3 Experiments on Benchmark Problems

#### 3.1 Experiments Setup

In order to test the fuzzy evaluation system, the benchmark data sets of Carter et al. [13] were used. The 12 instances that we studied, with different characteristics and various level of complexity, are shown in Table 1. Note that we are using the notation introduced in [24].

- Rule 1:** IF (*average penalty is low*) AND (*highest penalty is low*)  
THEN (*quality is very high*)
- Rule 2:** IF (*average penalty is low*) AND (*highest penalty is medium*)  
THEN (*quality is high*)
- Rule 3:** IF (*average penalty is low*) AND (*highest penalty is high*)  
THEN (*quality is medium*)
- Rule 4:** IF (*average penalty is medium*) AND (*highest penalty is low*)  
THEN (*quality is high*)
- Rule 5:** IF (*average penalty is medium*) AND (*highest penalty is medium*)  
THEN (*quality is medium*)
- Rule 6:** IF (*average penalty is medium*) AND (*highest penalty is high*)  
THEN (*quality is low*)
- Rule 7:** IF (*average penalty is high*) AND (*highest penalty is low*)  
THEN (*quality is medium*)
- Rule 8:** IF (*average penalty is high*) AND (*highest penalty is medium*)  
THEN (*quality is low*)
- Rule 9:** IF (*average penalty is high*) AND (*highest penalty is high*)  
THEN (*quality is very low*)

**Fig. 2.** Fuzzy rules for *Fuzzy Evaluation System*

**Table 1.** Examination timetabling problem characteristics

Data set	Number of slots ( $T$ )	Number of exams ( $N$ )	Number of students ( $S$ )
CAR-F-92I	32	543	18419
CAR-S-91I	35	682	16925
EAR-F-83I	24	190	1125
HEC-S-92I	18	81	2823
KFU-S-93	20	461	5349
LSE-F-91	18	381	2726
RYE-F-92	23	486	11483
STA-F-83I	13	139	611
TRE-S-92	23	261	4360
UTA-S-92I	35	622	21266
UTE-S-92	10	184	2750
YOR-F-83I	21	181	941

For each instance of the 12 data sets, 40 timetable solutions were constructed using a simple sequential constructive algorithm with backtracking, as previously implemented in [3]. We used eight different heuristics to construct the timetable solutions, for each of which the algorithm was run five times to obtain a range of solutions. However, due to the nature of the heuristics used, in some cases, a few of the constructed timetable solutions have the same proximity cost value. Therefore, for the purpose of standardisation, 35 different timetable solutions were selected out of the 40 constructed timetable solutions, by firstly removing any repeated solution instances and then just removing at random any excess. The idea is to obtain a set of timetable solutions with variations of timetable solution quality, in which none of the solutions have the same quality in terms of proximity cost (i.e average penalty per student). The timetable solutions were constructed by implementing the following heuristics:

- Three different single heuristic orderings:
  - Least Saturation Degree First (SD),
  - Largest Degree First (LD), and
  - Largest Enrollment First (LE),
- Three different fuzzy multiple heuristic orderings:
  - a *Fixed Fuzzy LD+LE Model*,
  - a *Tuned Fuzzy LD+LE Model*, and
  - a *Tuned Fuzzy SD+LE Model* (see [3] for details of these), and
- random ordering, and
- deliberately ‘poor’ ordering (see below).

A specific ‘poor’ heuristic was utilised in an attempt to purposely construct bad solutions. The idea was to attempt to determine the upper bound of solution

quality (in effect, though not formally, the ‘worst’ timetable for the given problem instance). Basically the method was to deliberately assign student exams in adjacent time slots. In order to construct bad solutions, LD was initially employed to order the exams. Next, the exams were sequentially selected from this ordered exams list, and assigned to the time slot that caused the highest proximity cost; this process continued until all the exams were scheduled.

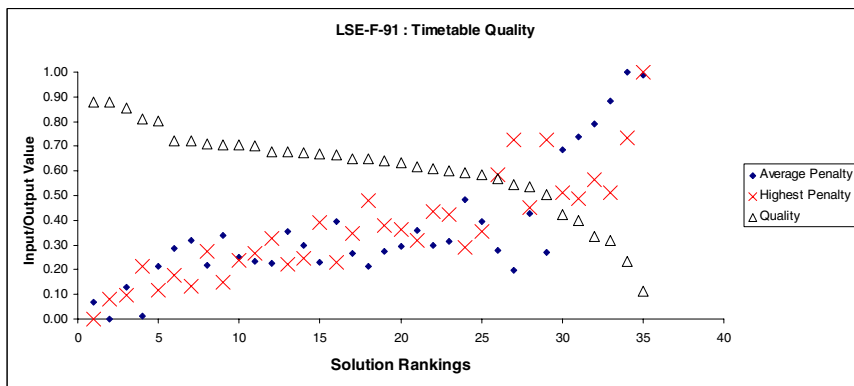
The 35 timetable solutions were analysed in order to determine the minimum and the maximum values for both the input variables, *average penalty* and *highest penalty*. These values were then used for the normalisation process (see Section 2.5). However, because the 12 data sets have various levels of complexity (see Table 1), the determination of the initial range for each data set is not a straightforward process. Thus, two alternative boundary settings were implemented in order to identify the appropriate set of *lowerBound* and *upperBound* for each data set. The first boundary setting used  $lowerBound = 0.0$  and the  $upperBound = maxValue$ , where *maxValue* is the largest value obtained from the set of 35 solutions. However, from the literature, the lowest value yet obtained for the STA-F-83I data set is around 130 [14]. Thus, it did not seem sensible to use zero as the lower bound in this case. In order to attempt to address this, we investigated the use of a non-zero lower bound. Of course, a formal method for determining the lower bound for any given timetabling instance is not currently known. Hence, the second boundary setting used  $lowerBound = minValue$  and  $upperBound = maxValue$ , where *minValue* is the smallest value obtained from the set of 35 constructed solutions for the respective data set.

In this implementation, both input variables, *average penalty* and *highest penalty*, were independently normalised based on their respective *minValue* and *maxValue*. The fuzzy evaluation system described earlier (see Section 2.4) was then employed to evaluate the timetable solutions. The same processes were applied to all of the data sets listed in Table 1. The fuzzy evaluation system was implemented using the ‘R’ language (*The R Foundation for Statistical Computing Version 2.2.0*) [25].

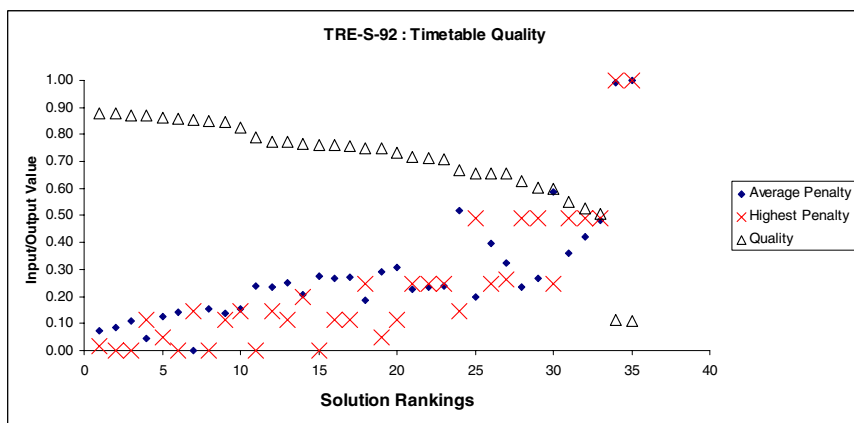
### 3.2 Experimental Results

In this section the experiment results are presented. Table 2 shows the minimum and maximum values obtained for both criteria. Figures 3(a) and 3(b) show the evaluation results obtained by the fuzzy evaluation system for the LSE-F-91 and TRE-S-92 data sets. These two data sets are shown as representative examples chosen at random. Both graphs show the results obtained when the boundary setting [*minValue*, *maxValue*] was implemented. In the graph, the *x*-axis (Solution Rankings) represents the ranking of the timetable solution quality evaluated by using the fuzzy evaluation function; in the order of the best solution to the worst solution. The *y*-axis represents the normalised input values (*average penalty* and *highest penalty*) and the output values (*quality*) obtained for the particular timetable solution. These two graphs show that the fuzzy evaluation





(a)



(b)

**Fig. 3.** Indicative illustrations of the range of normalised inputs and associated output obtained for the LSE-F-91 and TRE-S-92 data sets

function has performed as desired, in that the overall (fuzzy) quality of the solutions varies from close to zero to close to one.

Tables 3 and 4 show a comparison of the results obtained using the two alternative forms of the normalisation process. The *Solution Number* is used to identify a particular solution within the 35 timetable solutions used in the experiments for each data set. In both tables, the fifth and sixth columns (labeled as ‘Range [*minValue*, *maxValue*]’) indicate the fuzzy evaluation value and the rank of the solution relative to the other solutions, when the boundary range [*minValue*, *maxValue*] was used. The last two columns in the tables show the evaluation values and solution ranking obtained when the boundary range [0, *maxValue*] was used. Only the first 10 ‘best’ timetable solutions for each of

**Table 2.** Minimum and maximum values for *Average Penalty* and *Highest Penalty* obtained from the 35 timetable solutions for each data set

Data set	Average penalty		Highest penalty	
	Minimum value	Maximum value	Minimum value	Maximum value
CAR-F-92I	4.54	11.42	65.0	132.0
CAR-S-91I	5.29	13.33	68.0	164.0
EAR-F-83I	37.02	71.28	105.0	198.0
HEC-S-92I	11.78	31.88	75.0	136.0
KFU-S-93	15.81	43.40	98.0	191.0
LSE-F-91	12.09	32.38	78.0	191.0
RYE-F-92	10.38	36.71	87.0	191.0
STA-F-83I	160.75	194.53	227.0	284.0
TRE-S-92	8.67	17.25	68.0	129.0
UTA-S-92I	3.57	8.79	63.0	129.0
UTE-S-92	28.07	56.34	83.0	129.0
YOR-F-83I	39.80	64.48	228.0	331.0

the data sets are presented, based on the ranking produced when the boundary range [*minValue*, *maxValue*] was used.

## 4 Discussion

The fuzzy system presented here provides a mechanism to allow an overall decision in evaluating the quality of a timetable solution to be made based on common-sense rules that encapsulate the notion that the timetable solution quality increases as both the *average penalty* and the *highest penalty* decrease. The rules are in a form that is easily understandable by any timetabling officer.

Looking at Figure 3(a) and Figure 3(b) it can be seen that, in many cases, it is not guaranteed that timetable solutions with low *average penalty* will also have low *highest penalty*. This observation confirmed the assumption that considering only the proximity cost to measure timetable solution quality is not sufficient. As an example, if the detailed results obtained for the [0, *maxValue*] boundary range for LSE-F-91 in Table 3 are analysed, it can be seen that solution 13 (with the lowest *average penalty*) is not ranked as the ‘best’ solution. The same effect can be observed in solution 21 for the TRE-S-92 data set and solution 21 for the UTE-S-92 data set in Table 4.

In these three data sets (LSE-F-91, TRE-S-92 and UTE-S-92), the timetable solutions with the lowest *average penalty* were not selected as the ‘best’ timetable solution, because the decision made by the fuzzy evaluation system also takes into account another criterion, the *highest penalty*. This finding can also be seen in the other data sets, but it is not too obvious especially if we only focus on the first 3 ‘best’ solutions. Regardless, in terms of functionality, these results indicate

**Table 3.** A comparison of the results obtained using the two alternative forms of the normalisation process for six of the data sets

Data set	Timetable criteria			Range[minValue, maxValue]		Range[0, maxValue]	
	Solution number	Average penalty	Highest penalty	Evaluation value	Solution ranking	Evaluation value	Solution ranking
CAR-F-92I	19	4.544	65	0.888503	1	0.534427	1
	17	4.624	71	0.876804	2	0.517946	2
	18	4.639	71	0.876791	3	0.517485	3
	16	4.643	71	0.876788	4	0.517366	4
	7	5.148	68	0.876583	5	0.510084	5
	10	5.192	69	0.873279	6	0.506692	6
	13	5.508	68	0.858276	7	0.500729	7
	12	5.532	68	0.856617	8	0.500120	8
	11	5.595	68	0.851966	9	0.498538	9
	2	5.609	68	0.850863	10	0.498184	10
CAR-S-91I	17	5.292	68	0.888524	1	0.557585	1
	13*	5.573	75	0.880205	2	0.537593	3
	11*	5.911	68	0.879621	3	0.542750	2
	15	5.654	75	0.879244	4	0.535472	4
	14	5.842	75	0.875877	5	0.530812	5
	6*	6.079	76	0.868161	6	0.523516	8
	2*	6.393	71	0.860211	7	0.526116	6
	21*	6.509	71	0.853145	8	0.523572	7
	12	5.688	83	0.850233	9	0.520297	9
	16	5.690	83	0.850227	10	0.520255	10
EAR-F-83I	21	37.018	116	0.868135	1	0.467867	1
	4*	41.860	118	0.834883	2	0.444700	3
	5*	43.637	105	0.827016	3	0.454672	2
	18	44.147	118	0.798099	4	0.432416	4
	1	41.324	131	0.748303	5	0.415267	5
	3*	43.628	129	0.733864	6	0.411292	7
	20*	44.968	127	0.718542	7	0.411481	6
	12	49.662	114	0.710776	8	0.392966	8
	2*	41.178	144	0.699109	9	0.370814	11
	16*	44.980	135	0.674252	10	0.385906	9
HEC-S-92I	21	11.785	83	0.863057	1	0.506506	1
	14	14.774	75	0.854699	2	0.495547	2
	13	13.236	84	0.853706	3	0.489407	3
	7*	14.162	83	0.847966	4	0.482514	5
	16*	14.635	83	0.838633	5	0.477754	7
	15*	14.217	85	0.832653	6	0.476641	8
	1*	15.594	78	0.828916	7	0.481021	6
	6*	15.911	75	0.817611	8	0.485117	4
	27	15.763	84	0.801080	9	0.463727	9
	8*	14.124	94	0.727535	10	0.446459	11
KFU-S-93	17	15.813	98	0.888529	1	0.541211	1
	15	16.904	101	0.884358	2	0.526210	2
	14	17.336	100	0.883340	3	0.524294	3
	16	17.920	104	0.876034	4	0.513226	4
	3*	20.022	102	0.852341	5	0.501383	11
	9*	16.463	113	0.847871	6	0.509402	5
	7*	16.471	113	0.847868	7	0.509339	6
	6*	16.500	113	0.847858	8	0.509119	7
	8*	16.500	113	0.847858	9	0.509119	8
	10*	16.500	113	0.847858	10	0.509119	9
LSE-F-91	11*	13.458	78	0.881499	1	0.552817	2
	13*	12.094	87	0.879126	2	0.555747	1
	6*	14.720	89	0.855424	3	0.523229	4
	12*	12.349	102	0.812127	4	0.527563	3
	10*	16.408	91	0.804048	5	0.504874	5
	32*	17.942	98	0.722929	6	0.480142	7
	5*	18.564	93	0.720053	7	0.481747	6
	9*	16.486	109	0.707889	8	0.476028	9
	16*	18.979	95	0.707212	9	0.474395	11
	7*	17.174	105	0.704871	10	0.476479	8

**Table 4.** A comparison of the results obtained using the two alternative forms of the normalisation process for the remaining six data sets

Data set	Timetable criteria			Range[minValue, maxValue]		Range[0, maxValue]	
	Solution number	Average penalty	Highest penalty	Evaluation value	Solution ranking	Evaluation value	Solution ranking
RYE-F-92	21	10.384	87	0.888528	1	0.610225	1
	8	12.180	97	0.871582	2	0.558378	2
	10	12.337	97	0.870489	3	0.556102	3
	20	12.264	98	0.868672	4	0.555205	4
	6	12.976	97	0.864830	5	0.547756	5
	9	12.417	102	0.854386	6	0.545595	6
	7	12.094	105	0.839576	7	0.544225	7
	3*	13.678	104	0.831331	8	0.527428	12
	2*	14.441	104	0.817334	9	0.519821	14
	4*	14.581	104	0.814229	10	0.518513	15
STA-F-83I	21	160.746	227	0.888536	1	0.215426	1
	20	161.151	227	0.887829	2	0.214107	2
	15	164.375	228	0.871792	3	0.202156	3
	3	167.394	227	0.824391	4	0.196779	4
	31	168.195	227	0.805614	5	0.194967	5
	18	168.863	227	0.788882	6	0.193535	6
	11*	168.781	232	0.788385	7	0.182500	17
	16*	169.100	227	0.782864	8	0.193043	7
	29*	171.249	227	0.733062	9	0.188900	8
	9*	171.391	227	0.730410	10	0.188645	9
TRE-S-92	19*	9.311	69	0.880078	1	0.478231	2
	8*	9.389	68	0.878204	2	0.479078	1
	20	9.598	68	0.871588	3	0.475325	3
	7*	9.039	75	0.868946	4	0.468005	6
	6*	9.757	71	0.864316	5	0.465758	8
	17*	9.885	68	0.858365	6	0.469941	4
	21*	8.671	77	0.855435	7	0.469016	5
	1*	10.003	68	0.851293	8	0.467596	7
	10	9.856	75	0.846708	9	0.454514	9
	16*	9.981	77	0.826007	10	0.446743	11
UTA-S-92I	17	3.567	63	0.888536	1	0.532771	1
	11	3.833	68	0.878185	2	0.511100	2
	14	3.911	68	0.876019	3	0.508369	3
	13	3.927	68	0.875482	4	0.507798	4
	16	3.977	68	0.873738	5	0.506065	5
	12	4.143	68	0.866816	6	0.500466	6
	24	4.531	73	0.807693	7	0.475697	7
	23	4.573	73	0.802872	8	0.474319	8
	27	4.581	73	0.801938	9	0.474053	9
	8	4.976	68	0.762605	10	0.472232	10
UTE-S-92	19	30.323	83	0.879116	1	0.438284	1
	18	29.718	86	0.878651	2	0.429775	2
	21	28.069	90	0.853031	3	0.420748	3
	20	32.804	88	0.835146	4	0.400981	4
	26	31.522	91	0.826953	5	0.392480	5
	15	33.935	91	0.780095	6	0.378000	6
	27	34.928	90	0.767341	7	0.377994	7
	12*	32.996	94	0.758297	8	0.367082	9
	17*	29.695	98	0.723270	9	0.369027	8
	8	30.555	98	0.721926	10	0.362837	10
YOR-F-83I	21	39.801	234	0.883004	1	0.372139	1
	8*	44.158	233	0.837983	2	0.363036	3
	20*	44.412	231	0.831362	3	0.365581	2
	9	45.645	228	0.791749	4	0.359602	4
	14	45.736	238	0.785008	5	0.345675	5
	1	46.810	234	0.751639	6	0.341781	6
	2	46.862	235	0.749650	7	0.340088	7
	17	47.142	240	0.736830	8	0.330597	8
	32*	46.947	244	0.731929	9	0.324728	10
	31*	47.396	242	0.726141	10	0.324908	9

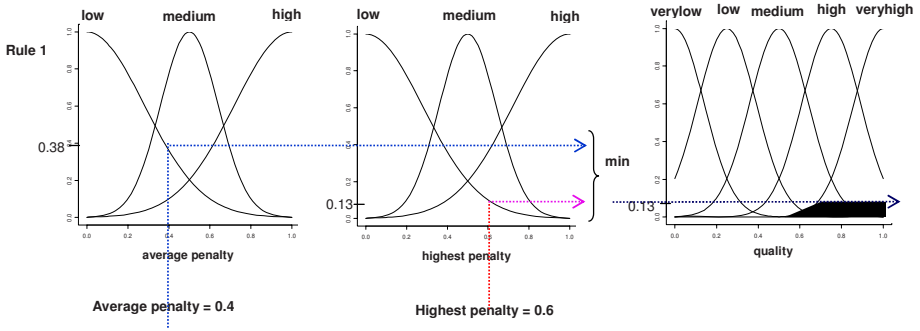
**Table 5.** Range of timetable quality

Data set	Range $[0, maxValue]$		Range $[minValue, maxValue]$	
	Worst solution	Best solution	Worst solution	Best solution
CAR-F-92I	0.111464	0.534427	0.111464	0.888503
CAR-S-91I	0.111464	0.557585	0.111464	0.888524
EAR-F-83I	0.111465	0.467867	0.111465	0.868135
HEC-S-92I	0.127502	0.506506	0.155374	0.863057
KFU-S-93	0.111466	0.541211	0.111466	0.888529
LSE-F-91	0.111895	0.555747	0.112182	0.881499
RYE-F-92	0.115999	0.610225	0.119240	0.888528
STA-F-83I	0.111464	0.215426	0.111464	0.888536
TRE-S-92	0.111476	0.479078	0.111488	0.880078
UTA-S-92I	0.111464	0.532771	0.111464	0.888536
UTE-S-92	0.111464	0.438284	0.111464	0.879116
YOR-F-83I	0.120046	0.372139	0.213388	0.883004

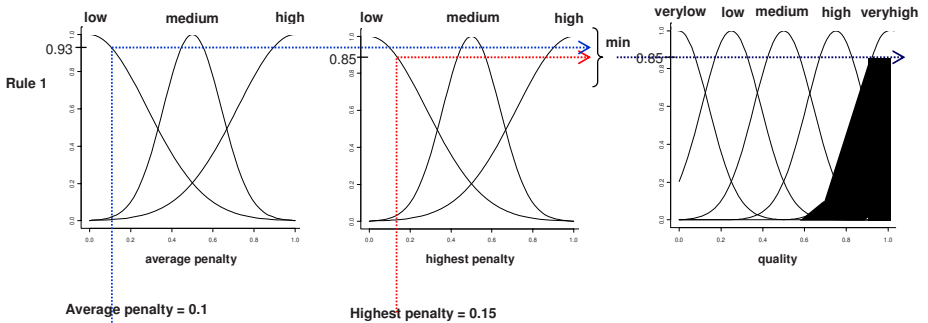
that the fuzzy evaluation system has performed as intended in measuring the timetable’s quality by considering two criteria simultaneously.

Analysing Tables 3 and 4 further, it can also be observed that the decision made by the fuzzy evaluation function is affected slightly when the different boundary settings are used to normalise the input values. The consequence of this is that the same timetable solution might be ranked in a different order, dependent on the boundary conditions. In both tables, the solutions with different ranking position are marked with \*. For the CAR-F-92I (in Table 3) and UTA-S-92I data sets (in Table 4), the solution rankings are unchanged by altering the boundary settings. In several cases, the solution rankings are only changed slightly. It is also interesting to note that, in a few cases, for example solution 3 for KFU-S-93 (in Table 3) and solution 11 for STA-F-83I (in Table 4), the ranking change is quite marked.

Overall, the performance of the fuzzy evaluation system utilising the boundary range  $[0.0, maxValue]$  did not seem as satisfactory as when the boundary range  $[minValue, maxValue]$  was used. This observation is highlighted by Table 5, which presents the fuzzy quality measure obtained for the ‘worst’ and ‘best’ solutions as evaluated under the two different boundary settings. When the boundary range  $[0.0, maxValue]$  was used, it can be seen that the fuzzy evaluation system evaluated the quality of the timetable solutions for the 12 data sets in the overall range of 0.111464 to 0.610225. In the case of STA-F-83I, the ‘best’ solution was only rated as 0.215426 in quality. The quality of timetable solutions falls only in the regions of linguistic terms that correspond to meanings of *very low*, *low* and *medium* in the timetable *quality* fuzzy set (see Figure 1(c)). This is because the lower bound value used here (i.e. *lowerBound* = 0.0) is far smaller than the actual smallest values. Consequently, the input values for even the lowest values (i.e. the ‘best’ solution qualities) are transformed to normalised



(a) Normalised value falls in the middle regions of the universe of discourse

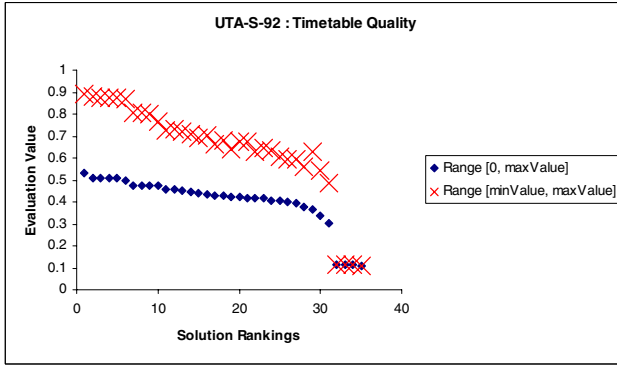


(b) Normalised value falls in the left regions of the universe of discourse

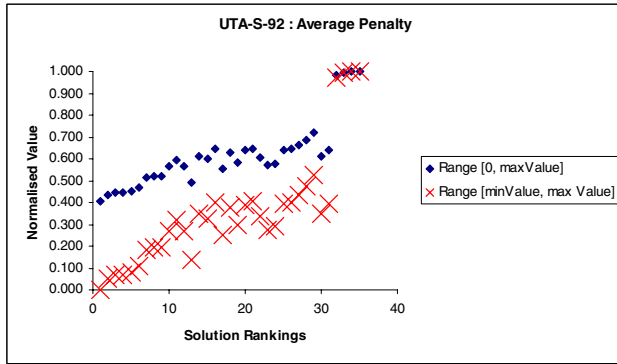
**Fig. 4.** Firing level for Rule 1 with different normalised input values

values that always fall within the regions of the *medium* and *high* linguistic terms in the input variables. As a result, the normalised input values will not cause any rule to be fired, or the firing level for any rule is relatively very low. This is illustrated in Figure 4(a), in which the activation level of the consequent part for Rule 1 is equal to 0.13. Although the possibility exists for any input to fall into more than one fuzzy set, so that more than one rule can be fired, the aggregation of fuzzy output for all rules will obtain a final shape that will only produce a low defuzzification value.

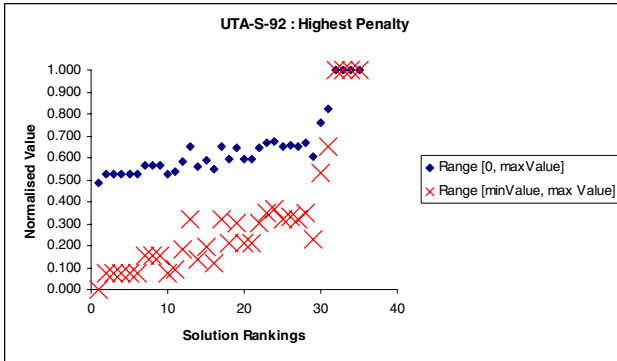
In contrast, Figure 4(b) illustrates the situation when the normalised input values fall in the regions of the linguistic terms that correspond to the meaning of *low*. In this situation, a high defuzzification value will be obtained due to the fact that most of the rules will have a high firing level. Thus, all of the solutions being ranked first had quality values more than 0.8, when the initial range [*minValue*, *maxValue*] was used. In this case, the quality of timetable solutions falls in the regions of the linguistic terms that correspond to meanings of *high* and *very high* for the timetable *quality* fuzzy set (see Figure 1(c)). As



(a)



(b)



(c)

**Fig. 5.** A graphical comparison of the effect of the two boundary settings for UTA-S-92I

might be expected, from the fact that the actual minimum and maximum values from the 35 constructed timetable solutions were used, the fuzzy evaluation results were nicely distributed along the universe of discourse of the timetable *quality* fuzzy set. For a clearer comparison of the effect of the two boundary settings, the distribution of input and output values for the UTA-S-92I data set are presented in Figure 5. As can be seen, the input values (Figure 5(b) and Figure 5(c)) are concentrated in the middle regions (0.4–0.7) of the graphs when the boundary range  $[0.0, \text{maxValue}]$  was used. In contrast, when the boundary range  $[\text{minValue}, \text{maxValue}]$  was used, the input values were concentrated in the bottom regions of the graphs. Based upon the defined fuzzy rules, we know that the timetable quality increases with a decrease in both input values. Indeed, this behaviour of the output can be observed for both boundary settings (see Figure 5(a)). Using either of the boundary settings, the fuzzy evaluation system is capable of ranking the timetable solutions. It is purely a matter of choosing the appropriate boundary settings of the fuzzy sets for the input variables. One of the deficiencies of this fuzzy evaluation, at present, appears to be that there is no simple way of selecting the boundary settings of the input variables. The drawback is that both boundary settings implemented so far can only be applied after a number of timetable solutions are generated. Therefore, significant amounts of times are required to construct and analyse the solutions. Furthermore, if boundary setting is based on the actual minimum and maximum values from the existing timetable solutions, the fuzzy evaluation system might not be able to evaluate a newly constructed timetable solution if the input values for the decision criteria for the new solution lie outside the range of the fuzzy sets. (Actually, output values *can* always be calculated – the real problem is that the resultant solution quality will always be the same once both criteria reach the left-hand boundary of their variables.) Thus it would be highly beneficial if we could determine approximate boundary settings, particularly some form of estimate of the lower bound of the assessment criteria, based upon the problem structure itself.

## 5 Conclusions

In conclusion, the experimental results presented here demonstrate the capability of a fuzzy approach of combining multiple decision criteria in evaluating the overall quality of a constructed timetable solution. However, in the fuzzy system implementation the selection of the *lowerBound* and *upperBound* for the normalisation process is extremely important because it has a significant effect on the overall quality obtained. The initial results presented here only use two decision criteria to evaluate the timetable quality. Possible directions for future research include extending the application of the fuzzy evaluation system by considering more criteria, and devising a more sophisticated approach to determine approximate boundary settings for the normalisation process. Another aspect to be investigated further is in comparing the quality assessments



produced by such fuzzy approaches with the subjective assessments of quality that timetabling officers make in real-world timetabling problems.

*Acknowledgments.* This research was supported by the Universiti Teknologi Malaysia (UTM) and the Ministry of Science, Technology and Innovation (MOSTI) Malaysia.

## References

1. Abdennadher, S., Marte, M.: University course timetabling using constraint handling rules. *Journal of Applied Artificial Intelligence* 14, 311–326 (2000)
2. Asmuni, H., Burke, E.K., Garibaldi, J.M.: A comparison of fuzzy and non-fuzzy ordering heuristics for examination timetabling. In: Lotfi, A. (ed.) *Proceedings of 5th International Conference on Recent Advances in Soft Computing*, pp. 288–293 (2004)
3. Asmuni, H., Burke, E.K., Garibaldi, J.M., McCollum, B.: Fuzzy multiple heuristic orderings for examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 334–353. Springer, Heidelberg (2005)
4. Boizumault, P., Delon, Y., Peridy, L.: Constraint logic programming for examination timetabling. *The Journal of Logic Programming* 26, 217–233 (1996)
5. Burke, E.K., Bykov, Y., Newall, J., Petrovic, S.: A time-predefined local search approach to exam timetabling problems. *IIE Transactions* 36, 509–528 (2004)
6. Burke, E.K., Elliman, D.G., Ford, P.H., Weare, R.F.: Examination timetabling in British universities – a survey. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 76–90. Springer, Heidelberg (1996)
7. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9, 451–470 (2003)
8. Burke, E.K., Newall, J.P., Weare, R.F.: A memetic algorithm for university exam timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996)
9. Burke, E.K., Newall, J.P.: A multistage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation* 3, 63–74 (1999)
10. Burke, E.K., Petrovic, S.: Recent research trends in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002)
11. Burke, E.K., Petrovic, S., Qu, R.: Case based heuristic selection for timetabling problems. *Journal of Scheduling* 9, 115–132 (2006)
12. Carter, M.W., Laporte, G.: Recent developments in practical exam timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996)
13. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society* 47, 373–383 (1996)
14. Casey, S., Thompson, J.: GRASPing the examination scheduling problem. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 232–244. Springer, Heidelberg (2003)
15. Cox, E., O'Hagen, M.: *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using and Maintaining Fuzzy Systems*. AP Professional, Cambridge, MA (1998)

16. Deris, S., Omatu, S., Ohta, H., Saad, P.: Incorporating constraint propagation in a genetic algorithm for university timetabling planning. *Engineering Applications of Artificial Intelligence* 12, 241–253 (1999)
17. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001)
18. Guéret, C., Jussien, N., Boizumault, P., Prins, C.: Building university timetables using constraint logic programming. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 130–145. Springer, Heidelberg (1996)
19. Kendall, G., Mohd Hussin, N.: A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA University of Technology. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 270–293. Springer, Heidelberg (2005)
20. Mamdani, E.H., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies* 7, 1–13 (1975)
21. Pappis, C., Siettos, C.: Fuzzy reasoning. In: Burke, E.K., Kendall, G. (eds.) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ch. 15, pp. 437–474. Springer, Berlin (2005)
22. Petrovic, S., Burke, E.K.: University timetabling. In: Leung, J. (ed.) *The Handbook of Scheduling: Algorithms, Models and Performance Analysis*, ch. 45, CRC Press, Boca Raton, FL (2004)
23. Petrovic, S., Patel, V., Yang, Y.: University timetabling with fuzzy constraints. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 313–333. Springer, Heidelberg (2005)
24. Qu, R., Burke, E.K., McCullom, B., Merlot, L.T.G., Lee, S.Y.: A survey of search methodologies and automated approaches for examination timetabling. *Computer Science Technical Report NOTTCS-TR-2006-4*, School of Computer Science and Information Technology, University of Nottingham (2006)
25. R Development Core Team: R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria (2005), ISBN 3-900051-07-0
26. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
27. Thompson, J.M., Dowsland, K.A.: A robust simulated annealing based examination timetabling system. *Computers and Operations Research* 25, 637–648 (1998)
28. Ueda, H., Ouchi, D., Takahashi, K., Miyahara, T.: Comparisons of genetic algorithms for timetabling problems. *Systems and Computers in Japan* 35, 1–12 (2004) [Translated from *Denshi Joho Tsushin Gakkai Ronbunshi*, J86-D-I, 691–701 (2003)]
29. White, G.M., Xie, B.S., Zonjic, S.: Using tabu search with longer-term memory and relaxation to create examination timetables. *European Journal of Operational Research* 153, 80–91 (2004)
30. Yang, Y., Petrovic, S.: A novel similarity measure for heuristic selection in examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 247–269. Springer, Heidelberg (2005)
31. Zadeh, L.A.: Fuzzy sets. *Information and Control* 8, 338–353 (1965)
32. Zimmermann, H.J.: *Fuzzy Set Theory and Its Applications*, 3rd edn. Kluwer, Dordrecht (1996)

# Linear Linkage Encoding in Grouping Problems: Applications on Graph Coloring and Timetabling

Özgür Ülker, Ender Özcan, and Emin Erkan Korkmaz

Department of Computer Engineering, Yeditepe University,  
Istanbul, Turkey

{oulker, eozcan, ekorkmaz}@cse.yeditepe.edu.tr  
<http://cse.yeditepe.edu.tr/ARTI>

**Abstract.** Linear Linkage Encoding (LLE) is a recently proposed representation scheme for evolutionary algorithms. This representation has been used only in data clustering. However, it is also suitable for grouping problems. In this paper, we investigate LLE on two grouping problems; graph coloring and exam timetabling. Two crossover operators suitable for LLE are proposed and compared to the existing ones. Initial results show that LLE is a viable candidate for grouping problems whenever appropriate genetic operators are used.

## 1 Introduction

In spite of the satisfactory performance of Evolutionary Algorithms (EA) on many NP optimization problems, the same achievement is not usually observed on grouping problems where the task is to partition a set of objects into disjoint sets. This is because the commonly used representations usually suffer from redundancies due to the ordering of groups. Moreover the genetic material might easily be disrupted by the genetic operators and/or by the rectification process after the operators are applied.

Timetabling problems are real-world NP-hard [7] problems. Discarding the rest of the constraints, attempting to minimize the timetabling slots while satisfying the clashing constraints turns out to be a graph coloring problem [19]. For this reason, new representation schemes and operators used in graph coloring are also of interest to the researchers in the timetabling community.

In the paper, we are investigating a recently proposed encoding scheme for grouping problems, Linear Linkage Encoding (LLE) [6]. LLE has only been tested on small clustering problem instances, and authors claim that the LLE performance is superior to Number Encoding (NE), the most common encoding scheme used in grouping problems. Unlike NE, LLE does not require an explicit bound on the number of groups that can be represented in a fixed-length chromosome. The greatest strength of LLE is that the search space is reduced considerably. There is a one-to-one correspondence between the chromosomes and the solutions when LLE is used. Consequently the aim of this paper is to present the

potential of the LLE representation on grouping problems. Previous studies denote that traditional crossover operators do not perform well. Therefore, a set of new crossover operators suitable for LLE are also tested on a set of problem instances including Carter's Benchmark [5] and DIMACS Challenge Suite [17].

This paper is organized as follows. We first define the grouping problems and common representations for them. The fundamentals of LLE is followed by the definition of the graph coloring problem. Then the operators of the algorithm with special crossovers are presented. Computational experiments and conclusions are given at the end of the paper.

## 2 Grouping Problems

Grouping problems [8] are generally concerned with partitioning a set  $V$  of items into a collection of mutually disjoint subsets  $V_i$  of  $V$  such that

$$V = V_1 \cup V_2 \cup V_3 \cup \dots \cup V_N \text{ and } V_i \cap V_j = \emptyset \text{ where } i \neq j.$$

Obviously, the aim of these problems is to partition the members of set  $V$  into  $N$  different groups where ( $1 \leq N \leq |V|$ ) each item is in exactly one group. In most of the grouping problems, not all possible groupings are permitted; a valid solution usually has to comply a set of constraints. For example in graph coloring, the vertices in the same group must not be adjacent in the graph. In the bin packing problem, the sum of the sizes of items of any group should not exceed the capacity of the bin, etc. Hence, the objective of grouping is to optimize a cost function defined over a set of valid groupings. In both graph coloring and bin packing the objective is to minimize the number of groups (independent sets and bins respectively) subjected to the mentioned constraints.

Grouping problems are characterized by the cost function based on the composition of the groups. An item in isolation has little or no meaning during the search process. Therefore, the building blocks that should be preserved in an evolutionary search should be the groups or the group segments.

### 2.1 Representations in Grouping Problems

The most predominant representation in grouping problems in both evolutionary and local search methods is Number Encoding (NE). In NE, each object is encoded with a group id indicating which group it belongs to. For example the individual 2342123 encodes the solution where first object is in group 2, second in 3, third in 4, and so on. However, it is easy to see that the encoding 1231412 represents exactly the same solution, since the naming or the ordering of the partition sets is irrelevant. The drawbacks of this representation are presented in [8] and it is pointed out that this encoding is against the minimal redundancy principles for encoding scheme [23].

Another representation for grouping problems is Group Encoding (GE). The objects which are in the same group are placed into the same partition set.

For instance, the above sequence can be represented as  $(1, 4, 6)(2, 7)(3)(5)$ . The ordering within each partition set is unimportant, since search operators work on groups rather than objects unlike in NE. However, the ordering redundancy among groups still holds. For instance,  $(2, 7)(3)(5)(1, 4, 6)$  would again represent the same solution.

### 2.2 Linear Linkage Encoding

LLE can be implemented using an array. Let the entries in the chromosome be indexed with values from 1 to  $n$ . Each entry in the array then holds one integer value which is a link from one object to another object of the same partition set. With  $n$  objects, any partition set on them can be represented as an array of length  $n$ . Two objects are in the same partition set if either one can be reached from another through the links. If an entry is equal to its own index, then it is considered as an ending node. The links in LLE are unidirectional, thus; backward links are not allowed. In short, in order to be considered as a valid LLE array, the chromosome should follow the following two rules:

- The integer value in each entry is greater than or equal to its index but less than or equal to  $n$ .
- No two entries in the array can have the same value; the index of an ending node is the only exception to this rule.

In LLE, the items in a group construct a linear path ending with a self referencing last item. It can be represented by the *labeled oriented pseudo (LOP) graph*. A LOP Graph is a labeled directed graph  $G(V, E)$ , where  $V$  is the vertex set and  $E$  is the edge set. A composition of  $G$  is a grouping of  $V(G)$  into disjoint oriented pseudo-path graphs  $G_1, G_2, \dots, G_m$  with the following properties:

- Disjoint paths:  $\bigcup_{i=1}^m V(G_i) = V(G)$  and for  $i \neq j, V(G_i) \cap V(G_j) = \emptyset$
- Non-backward oriented edges: If there is an edge  $e$  directed from vertex  $v_i$  to  $v_k$  then  $i \leq k$ .
- Balanced connectivity
  - $|E(G)| = |V(G)|$
  - each  $G_i$  has only one ending node whose *in-degree* = 2 and *out-degree* = 1
  - each  $G_i$  has only one starting node whose *in-degree* = 0 and *out-degree* = 1
- All other  $|V(G_i)| - 2$  vertices in  $G_i$  have *in-degree* = *out-degree* = 1.

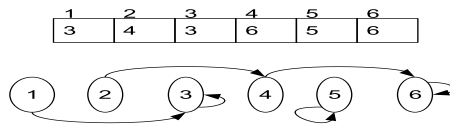


Fig. 1. LLE array and LOP graphs

There are three clear observations regarding LOP graphs:

1. Given a set of items  $S$ , there is one and only one composition of LOP graphs  $G(V, E)$  for each grouping of  $S$ , where  $|V| = |S|$ .
2. The number of LOP graphs is given by the  $n$ th Bell number [6].
3. LLE in array form is a unique implementation of the LOP graph.

### 2.3 Exam Timetabling as a Grouping Problem

Exam timetabling requires satisfactory assignment of timetable slots (periods) to a set of exams. Each exam is taken by a number of students, based on a set of constraints. In most of the studies, NE-like representations are used. In [3], a randomly selected light or a heavy mutation followed by a hill climbing method was applied. Various combinations of constraint satisfaction techniques with genetic algorithms can be found in [25]. Paquete et al. [22] applied a multi-objective evolutionary algorithm based on Pareto ranking with two objectives: minimize the number of conflicts within the same group and between groups. Wong et al. [26] applied a GA with a non-elitist replacement strategy. After genetic operators are applied, violations are repaired with a hill climbing fixing process. In their experiments a single problem instance was used. Ozcan et al. [21] proposed a memetic algorithm (MA) for solving exam timetabling at Yeditepe University. MA utilizes a violation directed adaptive hill climber.

Considering the task of minimizing the number of exam periods and removing the clashes, exam timetabling reduces to the graph coloring problem [19].

### 2.4 Graph Coloring Problem as a Grouping Problem

The Graph Coloring Problem (GCP) is a well known combinatorial optimization problem which is proved to be NP-complete [11]. Informally stated, graph coloring is assigning colors to each vertex of an undirected graph such that no adjacent vertices should receive the same color. The minimal number of colors that can be used for a valid coloring is called the chromatic number. A more formal definition is as follows.

Given a graph  $G = (V, E)$  with vertex set  $V$  and edge set  $E$ , and given an integer  $k$ , a  $k$ -coloring of  $G$  is a function  $c : V \rightarrow 1, \dots, k$ . The value  $c(x)$  of a vertex  $x$  is called the color of  $x$ . The vertices with color  $r$  ( $1 \leq r \leq k$ ) define a color class, denoted  $V_r$ . If two adjacent vertices  $x$  and  $y$  have the same color  $r$ ,  $x$  and  $y$  are conflicting vertices, and the edge  $(x, y)$  is called a conflicting edge. If there is no conflicting edge, then the color classes are all independent sets and the  $k$ -coloring is valid. The GCP is to determine the minimum integer  $k$  (the chromatic number of  $G - \chi(G)$ ) such that there exists a legal  $k$ -coloring of  $G$  [1].

In the literature there are many heuristics devised for finding chromatic number and solving  $k$ -coloring problems. Early applications of GCP solvers are simple constructive methods [2,19] which color each vertex of the graph one after another based on dynamic ordering of the vertices according to its saturation

degree. Local search methods such as tabu search [14] and simulated annealing [16] have been followed with hybridizations of these techniques with genetic algorithms [9,10] which resulted in the state of the art graph coloring algorithms.

Graph coloring is generally considered as a difficult problem for pure genetic algorithms [13]. Currently, the most successful algorithms are memetic algorithms [9,10] which hybridize the evolutionary techniques with a local search method. In this approach, the role of genetic operators is limited to finding promising points in the search landscape from which the local search can initiate. Hence, the exploration of the search space is carried out by the local search operator. For instance in Galinier and Hao's hybrid algorithm [10], a crossover operation is preceded by a tabu search procedure which may last thousands of tabu iterations.

There are mainly two reasons for the unsuccessful attempts of using pure genetic implementations on graph coloring: the redundancies inherent in the representations used for the encoding of the chromosome, and lack of a suitable crossover operator which would transmit the building blocks efficiently, preferably with some domain knowledge. In this paper, we are mainly interested in the representational issues, but we also propose suitable crossover operators for the proposed multi-objective genetic algorithm.

### 3 A Multi-objective Genetic Algorithm for Graph Coloring and Timetabling

Note that our main intention in this study is to propose a multi-objective solution foundation to multi-constraint timetabling problems. To our knowledge, none of the efficient graph coloring algorithms in the literature empowers genetic operators as their main search mechanism. These methods usually rely on local search operators. We are more interested in the applicability of linear linkage encoding on grouping problems by using suitable crossover and mutation operators. We present a multi-objective genetic algorithm employing weak elitism and the main search operator of this approach is mutation aided by crossover.

#### 3.1 Initialization

Since we are dealing with a minimal coloring problem (where the objectives are to minimize the number of colors and number of conflicting edges), it is desirable to initialize the population with individuals having different number of colors. Setting the range of number of colors too wide will unnecessarily increase the search space and thus the execution time. It is also undesirable to set the range too narrow either. Such a scheme will prevent promising individuals with different number of colors from co-operating through crossover and mutation. Tight lower and upper bounds can be found based on the maximal clique and maximal degree of the graph. Fast heuristics or approximation algorithms such as RLF [19] or DSATUR [2] can be used to find these bounds as well. Since exact or approximate chromatic numbers in the test instances are already known, these bounds are set manually in this study.

In our experiments, we have used a population with individuals having different number of colors and an external population which holds the best individuals with the minimal conflicts for a specific number of colors within a search range ( $lowerBound \leq k \leq upperBound$ ). In order to create an individual, first  $k$  is determined, then a  $k$ -colored individual is randomly created. An external smart initialization method was not used to reduce the edge conflicts in order not to give any bias to our crossover operators and let the multi-objective evolutionary method do the search.

### 3.2 Selection

A  $k$ -coloring problem is solved when the number of conflicting edges is zero. If a  $k$ -coloring solution is obtained,  $k + 1$  colorings can also be generated by dividing independent sets into two. It might be possible to unite two sets in a  $k + 1$  coloring to obtain a  $k$ -coloring. The Pareto front will almost be a straight line along the color axis with zero conflict if the lower bound is set close to the chromatic number. A restricted multi-objective method might work efficiently on a search range within specified bounds around the chromatic number.

As a multi-objective genetic algorithm a modified version of Niche Genetic Algorithm (NPGA) described in [15] was used. In NPGA, two candidate individuals are selected at random from the population to be one of the parents. A comparison set is formed from randomly selected individuals within the population. Each candidate is then compared against each individual in the comparison set. If one candidate is dominated by the comparison set (which means it is worse for every part of the objective function than any individual in the comparison set) and the other is not, then the latter is selected for reproduction. If neither or both are dominated by the comparison set, then niching is used to select a winner mate. The size of comparison set ( $t_{dom}$ ) allows a control over the selection pressure. The comparison set size was preset to around ten percent of the population size as suggested by [15].

When neither or both candidates are dominated by the comparison set, the candidate with a smaller niche count is selected for reproduction. We calculate the niche value  $m_i$  of the  $i$ th individual by

$$m_i = \sum_{j \in pop} sh(d[i, j]) \quad (1)$$

where  $d[i, j]$  is the distance between two individuals according to their objective function values and  $sh(d)$  is the sharing function which is

$$sh(d) = \begin{cases} 1 & \text{if } d = 0 \\ 1 - d/\mu_{share} & \text{if } d < \mu_{share} \\ 0 & \text{if } d \geq \mu_{share} \end{cases} \quad (2)$$

and the distance measure is the Manhattan distance in terms of color and conflict values in the individuals. The objective functions  $c_{ix}$  and  $c_{jx}$  represent the number of colors and edge conflicts respectively for parents  $i, j$  where  $x = \{1, 2\}$ :

$$d[i, j] = |c_{i1} - c_{j1}| + |c_{i2} - c_{j2}|. \quad (3)$$



### 3.3 Redundancy and Genetic Operators

Although LLE in theory is a non-redundant representation for grouping problems, practically this advantage disappears if the search operators do not adhere to this principle. Therefore a more desirable option is to make the search non-redundant additional to the representation. For example consider a basic hill climbing mutation which sends one vertex from one set to another. This is analogous of changing a gene value in the number encoding. If the majority of the group ids of the items can be maintained for a long period of time, then it is quite possible to make a low-redundant search even on a highly redundant encoding such as NE. This is one of the reasons local search based methods are quite successful on grouping problems. Because of the small perturbations on the search space, these methods not only preserve the building blocks on the candidate solution but also are able to operate on a low-redundant small region of the large search landscape.

The same advantage, unfortunately, does not hold for crossover which makes huge jumps on the search space. It is possible to keep the majority of the group ids of the items fixed by using traditional crossovers like one-point or uniform crossover. Such methods, however, do not preserve the groups which are the building blocks themselves. Therefore, a crossover operator should preserve the order of the colors as long as possible. Two ordering mechanism which assigns group ids to the groups after crossover and mutation are investigated within the context of LLE. These two redundancy elimination mechanisms are based on the cardinality of the groups and the lowest index number at each group. In [24], the authors investigated the effect of these two methods on Graph Coloring by using 0/1 ILP SAT solvers.

**Cardinality-Based Ordering.** In cardinality-based ordering, each group receives a group id according to its cardinality (set size). Groups are sorted according to their cardinality and the group with the highest cardinality will be assigned group id 1, the second highest will be identified as group 2, and so on. For example groups  $(1, 3)(5)(2, 4, 6)$  are indexed as  $V_1 = (2, 4, 6)$ ,  $V_2 = (1, 3)$ , and  $V_3 = (5)$ . Since more than one group can have the same cardinality, the ordering might not be unique.

**Lowest Index Ordering.** In lowest index ordering, the smallest index in each group is found first, then the group with the smallest index number is assigned group id 1, the group with the second smallest index number is assigned group id 2, and so on. For example, groups  $(1, 3)(5)(2, 4, 6)$  are indexed as  $V_1 = (1, 3)$ ,  $V_2 = (2, 4, 6)$ , and  $V_3 = (5)$ . Since each group has one unique lowest index, the ordering is always unique.

### 3.4 Crossover

LLE can be implemented using one dimensional arrays, allowing applicability of the traditional crossover methods such as, one point or uniform crossover. However, it is observed that these crossovers can be too destructive especially

for graph coloring due to the danger of introducing new links in the LOP graph absent in both parents. Also, since the building blocks [12] in graph coloring are strictly large independent sets (not even independent set segments), there is a risk of destroying these building blocks. However, for small problem instances, one-point crossover in LLE is reported to generate satisfactory results for the clustering problem [6]. (This might be due to the fact that building blocks may be a segment of clusters rather than the whole cluster.)

Unfortunately we have observed a very poor performance from one-point crossover in our experiments. It was not even able to generate solutions in the color search range we specified.

Three types of crossover operators are compared using LLE representation.

**Greedy Partition Crossover.** The GCP can be considered as partitioning the graph into independent sets. Therefore by preserving the large independent sets, the vertices in non-independent sets can be forced to form independent sets as well.

---

**Algorithm 1.** Greedy Partition Crossover

---

**Require:** Two Parents – *parent1* and *parent2* in LLE form.

**Ensure:** One *offspring* in LLE form.

```

1: currentParent = Random(parent1, parent2).
2: repeat
3:   largestSet = Find largest set in currentParent.
4:   transmit unassigned the vertices (links) in the largestSet to offspring.
5:   mark transmitted vertices as assigned.
6:   if currentParent = parent1 then
7:     currentParent = parent2.
8:   else
9:     currentParent = parent1.
10:  end if
11: until all vertices are assigned
12: if Lowest Index Ordering is Used then
13:   sort group ids according to the lowest index number (GPX-LI).
14: else
15:   sort group ids according to the cardinality (GPX-CB).
16: end if

```

---

Greedy Partition Crossover (GPX) was proposed by Galinier and Hao [10] in their Hybrid Graph Coloring Algorithm. The idea is to transmit the largest set (group) from one parent, then to delete the vertices in this largest set from the other parent. This transmission and deletion process is repeated on both parents successively until all of the vertices are assigned to the child.

Two forms of GPX by following the rules of Cardinality and Lowest Index Ordering are implemented. The difference is just assigning the color ids to the groups after the crossover. In GPX Lowest Index Crossover (GPX-LI), the groups with lower index numbers are given lower color ids, whereas in GPX Cardinality

Based Crossover (GPX-CB), the lower color ids are assigned to the groups with higher cardinality. A general pseudocode of GPX is represented in Algorithm 1.

Consider two parents in Figure 2. We can obtain the child as follows: Largest Set in parent 1 is (3, 4, 5, 6). This set is transmitted to the child and 3, 4, 5 and 6 are deleted from parent 2. After this deletion the largest set in parent 2 (1) is transmitted to the child. Finally (2) is assigned as the last group. After sorting according to the lowest index ordering (GPX-LI), the coloring then becomes  $C_1 = (1)$ ,  $C_2 = (2)$ ,  $C_3 = (3, 4, 5, 6)$ . If the groups are sorted according to their cardinality (GPX-CB), the coloring is  $C_1 = (3, 4, 5, 6)$ ,  $C_2 = (1)$ ,  $C_3 = (2)$ .

Both GPX-LI and GPX-CB are applicable to other representations such as number or group encodings. Our intention of using these crossovers is to create crossover operators applicable only to LLE. The following two crossovers are inspired from GPX.

**Lowest Index First Crossover.** In Lowest Index First Crossover (LIFX), the goal is to transmit the groups beginning with lowest index numbers. LIFX works as follows.

---

**Algorithm 2.** Lowest Index First Crossover

---

**Require:** Two Parents - *parent1* and *parent2* in LLE form.

**Ensure:** One *offspring* in LLE form.

- 1:  $i = 1$
  - 2:  $currentParent = \text{Random}(parent1, parent2)$ .
  - 3: **repeat**
  - 4:  $lengthOfParent = \text{Calculate the path length of } currentParent \text{ starting from } i$ .
  - 5: transmit unassigned vertices (links) in the *parentToSelect* to *offspring*.
  - 6: mark transmitted vertices as assigned.
  - 7:  $i = \text{next unassigned vertex}$ .
  - 8: **if**  $currentParent = parent1$  **then**
  - 9:      $currentParent = parent2$ .
  - 10: **else**
  - 11:      $currentParent = parent1$ .
  - 12: **end if**
  - 13: **until** all vertices are assigned
- 

A parent is randomly selected. Beginning with the lowest index (vertex) which has not been assigned yet, the vertices are transmitted to the child by following the links. If the vertices along the path are assigned before, they are skipped. The process is repeated by successively changing the parents for transmission until all of the vertices are assigned to the child. A general pseudocode of LIFX is represented in Algorithm 2.

The application of LIFX on the parents in Figure 2 would be as follows: Assuming we begin with the first parent, current lowest index number is 1. Therefore, (1, 2) is transmitted to the child. The current lowest index number is now 3. Switching to parent 2, we copy (3, 6) as the next group. Switching back

to parent 1, current lowest index is 4, therefore (4, 5) is copied to the child. Final coloring then becomes:  $C_1 = (1, 2)$ ,  $C_2 = (3, 6)$ ,  $C_3 = (4, 5)$ .

Note that this crossover prioritizes groups beginning with the lowest index number, therefore it reduces the sizes of the groups beginning with higher index numbers. This is in concordance with the nature of LLE, because the number of possible values for the higher index locations is lower.

**Lowest Index Max Crossover.** In Lowest Index Max Crossover (LIMX), the child is generated with two objectives: Transmit large groups to preserve Cardinality Based Ordering, and to transmit groups beginning with lowest index number (to preserve Lowest Index Ordering). Therefore this method can be considered as an amalgamation of LIFX and GPX.

---

**Algorithm 3.** Lowest Index Max Crossover

---

**Require:** Two Parents – *parent1* and *parent2* in LLE form.

**Ensure:** One *offspring* in LLE form.

```

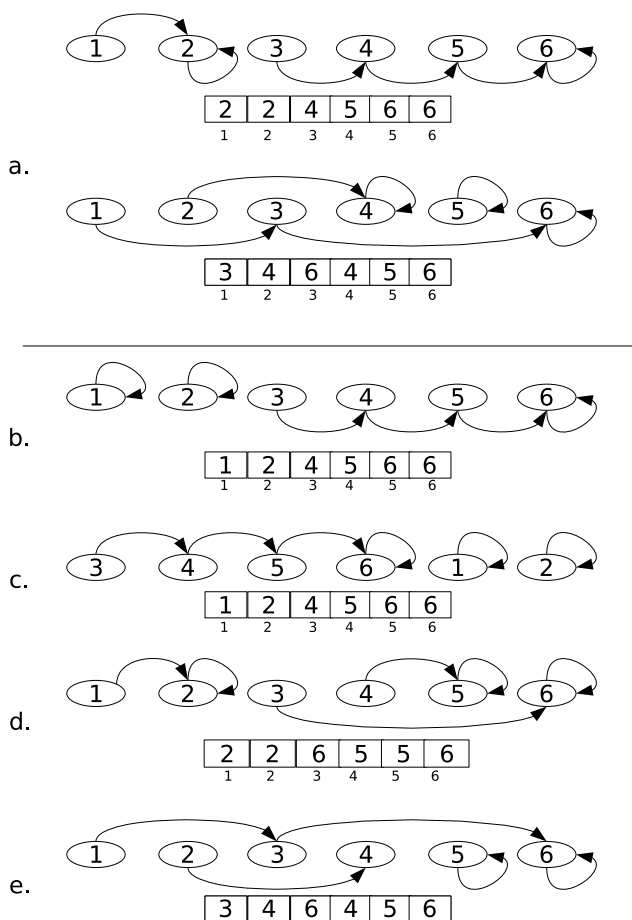
1:  $i = 1$ 
2: repeat
3:    $lengthOfParent1 =$  Calculate the path length of parent1 starting from  $i$ .
4:    $lengthOfParent2 =$  Calculate the path length of Parent1 starting from  $i$ .
5:   if  $LengthOfParent1 < LengthOfParent2$  then
6:      $parentToSelect = parent1$ .
7:   else
8:      $parentToSelect = parent2$ .
9:   end if
10:  transmit unassigned vertices (links) in the parentToSelect to offspring.
11:  mark transmitted vertices as assigned.
12:   $i =$  next unassigned vertex.
13: until all vertices are assigned

```

---

LIMX works as follows. Beginning with the lowest index number (vertex) which has not been assigned first we calculate the length of the links (path length) in both parents. Already assigned vertices are not counted in this link length calculation. This allows finding the largest set in parents beginning with the lowest index number. Then the links (and thus vertices) are transmitted to the child from the parent with the greater link-length. After that next unassigned lowest index number is found and the process is repeated until all vertices are assigned. A general pseudocode of LIMX is represented in Algorithm 3.

Application of LIMX to parents in Figure 2 is as follows: Current lowest index is 1. (1, 3, 6) is longer than (1, 2) so (1, 3, 6) is copied to the child. Current lowest index is now 2. (2, 4) is larger than (2) so it is transmitted to the child. Finally (5) is copied to the child as the last group. At the end of LIMX the coloring then becomes:  $C_1 = (1, 3, 6)$ ,  $C_2 = (2, 4)$ ,  $C_3 = (5)$ .



**Fig. 2.** (a) Two Parents in LLE Array and LOP Graph form. (b) Resulting offspring from Greedy Partition Crossover – Lowest Index Ordering. (c) Resulting offspring from Greedy Partition Crossover – Cardinality-Based Ordering. (d) Resulting offspring from Lowest Index First Crossover. (e) Resulting offspring from Lowest Index Max Crossover.

### 3.5 Mutation

We have used a mutation scheme that sends a selected conflicting vertex  $x$  from its color set to the best possible other one. A tournament method is used to select a vertex for transfer. A percentage of conflicting vertices are taken into a tournament and the vertex with the highest conflict in this set is transferred to a best color available.

**Table 1.** Test setup

Test machine	Pentium 4 2Ghz with 256MB Ram
Compiler	GCC C++ 3.2 with -O2 flags
No of generations	10000
Population size	25 percent of the number of vertices in graph
Comparison set size	10 percent of the population size
Niche size	5.0
Crossover rate	0.25
Mutation rate	a single mutation is enforced
Number of runs	50 for each instance

As aforementioned, assigning group ids after crossover is essential for low redundancy and the success of the mutation. In GPX-LI, LIMX and LIFX, the ids are assigned according to Lowest Index Ordering whereas in GPX-CB the ids are assigned according to Cardinality-Based Ordering.

### 3.6 Replacement

In our simulations we have employed a trans-generational replacement with weak elitism. At each generation,  $\lambda$  (non elitist) +  $\mu$  (elitist individuals, one for each number of colors within the searching range) individuals produce  $\lambda$  children. If new best individuals for each color are found in the new children, they are moved to the population with elitist individuals. The remaining children form the next generation.

## 4 Experiments

In our tests, we use several graphs from the DIMACS Challenge Suite [17]. The general test setup is summarized in Table 1.

In Table 2, we represent the characteristics of the test instances sampled from the DIMACS test suite. Table shows the name, number of vertices ( $|V|$ ), number of edges ( $|E|$ ), edge density (%) and chromatic number ( $\chi(G)$ ) of the instances.

In all our tests, the mutation count is set to 1, and crossover rate is fixed at 0.25. In this setup, the algorithm is more like a genetic hill climbing method. Since the chromatic number of these graphs are already known, we have set the range by hand according to the chromatic number  $\chi(G)$ .

Note that our primary intention is to compare the crossover operators in the context of LLE. As a result, we did not run our experiments for a long time. (The longest time required for one run is around 5 minutes for cars91 graph instance). This might have resulted in performance hit for large problem instances which may need an exponential increase rather than a linear increase in the maximum number of generation.

In Table 3, we present the best solutions obtained after 50 runs by using the four crossover operators mentioned. Figure 3 represents the average color number of 50 runs for some of the instances in DIMACS suite. The results show

**Table 2.** Data characteristics about the problem instances from the DIMACS Suite

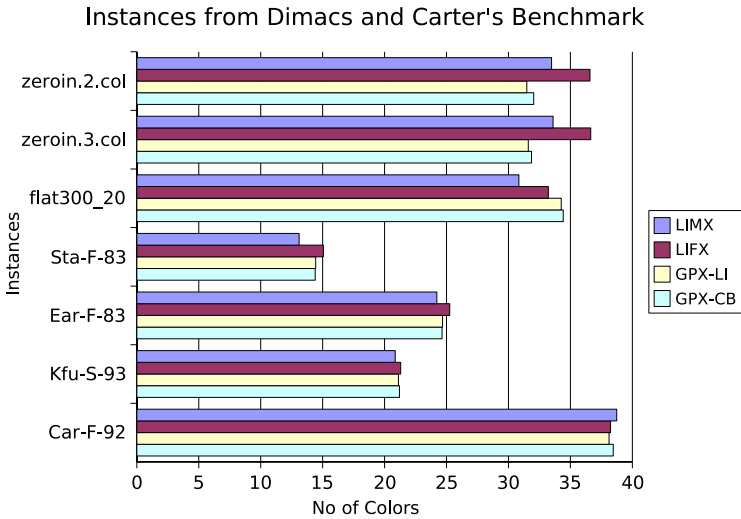
Instance	$ V $	$ E $	%	$\chi(G)$
DSJC125.5	125	3891	0,50	?
DSJC125.9	125	6961	0,90	?
zeroin.1.col	211	4100	0,19	49
zeroin.2.col	211	3541	0,16	30
zeroin.3.col	206	3540	0,17	30
DSJC250.1	250	3218	0,10	?
DSJC250.5	250	15668	0,50	?
DSJC250.9	250	27897	0,90	?
flat300_20	300	21375	0,48	20
flat300_26	300	21633	0,48	26
flat300_28	300	21695	0,48	28
school1_nsh	352	14612	0,24	14
le450_15a	450	8168	0,08	15
le450_15b	450	8169	0,08	15
le450_15c	450	16680	0,17	15
le450_15d	450	16750	0,17	15
le450_25a	450	8260	0,08	25
le450_25b	450	8263	0,08	25
le450_25c	450	16680	0,17	25
le450_25d	450	16750	0,17	25
DSJC500.1	500	12458	0,10	?
DSJC500.5	500	62624	0,50	?

no significant statistical differences between crossover operators except for a few instances. For example for flat300\_20 graph, LIMX was able to find a best 20 coloring while the other crossovers were very far from the optimal. However, for this graph, average colorings found with all crossovers and standard deviation are quite high. This is possibly due to the natural difficulty of flat graphs. Another slight difference appeared in register allocation graphs (zeroin.X.col graphs) where LIFX performed worst while GPX crossovers performed best.

We have also presented the graph coloring algorithm results of Kirovski and Potkonjak [18] for two set of parameters (Kirovski B and Kirovski C). Kirovski and Potkonjak’s algorithm is based on divide-and-conquer paradigms, global search for constrained independent sets, assignment of most-constrained vertices to least constraining colors, reuse and locality exploration of intermediate solutions, and post-processing lottery-scheduling iterative improvement. With respect to Kirovski and Potkonjak’s solutions, our crossovers gave similar, and for some instances better, results; however, when the instance becomes larger and more difficult, Kirovski and Potkonjak’s algorithm performs better. However, our primary intention was not to compare LLE representation with state of the art algorithms but to compare the crossover operators as stated before.

**Table 3.** Best colorings obtained for the instances in the DIMACS Benchmark Suite

Instance	$\chi(G)$	LIMX	LIFX	GPX-LI	GPX-CB	Kirovski-B	Kirovski-C
DSJC125.5	?	18	18	18	18	19	18
DSJC125.9	?	44	44	44	44	45	45
zeroin.1.col	49	49	50	49	49	49	49
zeroin.2.col	30	31	35	31	31	30	30
zeroin.3.col	30	31	35	30	31	30	30
DSJC250.1	?	9	9	9	9	9	9
DSJC250.5	?	31	31	31	31	30	30
DSJC250.9	?	75	75	75	74	77	77
flat300_20	20	20	31	27	32	20	20
flat300_26	26	34	34	34	34	32	28
flat300_28	28	34	34	34	34	33	32
school1_nsh	14	14	14	14	14	16	14
le450_15a	15	16	16	16	16	17	17
le450_15b	15	16	16	16	16	17	17
le450_15c	15	23	23	23	23	22	21
le450_15d	15	23	23	23	23	22	21
le450_25a	25	25	25	25	25	25	25
le450_25b	25	25	25	25	25	25	25
le450_25c	25	28	29	28	28	28	28
le450_25d	25	28	28	28	28	?	?
DSJC500.1	?	14	14	14	14	14	14
DSJC500.5	?	55	55	55	55	51	50



**Fig. 3.** Average number of colors (groups) for some instances in DIMACS and Carter's Benchmark



**Table 4.** Data characteristics of the problem instances from the Carter Benchmark Suite

Instance	$ V $	$ E $	%
Hecs92	81	1363	0.42
Staf83	139	1381	0.14
Yorf83	181	4691	0.29
Utes92	184	1430	0.08
Earf83	190	4793	0.27
Tres92	261	6131	0.18
Lsef91	381	4531	0.06
Kfus93	461	5893	0.06
Ryes93	486	8872	0.08
Carf92	543	20305	0.14
Utas92	622	24249	0.13
Cars91	682	29814	0.13

**Table 5.** Best colorings obtained for the instances in the Carter Benchmark Suite

Instance	LIMX	LIFX	GPX-LI	GPX-CB	Carter	Caramia	Merlot
Hecs92	17	17	17	17	17	17	18
Staf83	13	14	14	14	13	13	13
Yorf83	20	20	20	20	19	19	23
Utes92	10	10	10	10	10	10	11
Earf83	23	24	24	23	22	22	24
Tres92	21	21	21	21	20	20	21
Lsef91	17	18	18	18	17	17	18
Kfus93	20	20	20	20	19	19	21
Ryes93	23	23	23	23	21	21	22
Carf92	36	36	36	36	28	28	31
Utas92	38	39	38	38	32	30	32
Cars91	36	36	37	35	28	28	30

Table 4 represents some instances taken from the Carter’s Benchmark [5]. We again represent the number of vertices, edges and edge density of these graphs in this table. Table 5 represents the best colorings obtained after 50 runs. In Figure 3, the average colorings of 50 runs for some instances in Carter’s benchmark are represented.

For the instances in the Carter’s timetabling benchmark, again, a significant difference among crossover operators is not observed. However, LIMX has a slightly better performance in terms of best and average color (group) number. LIMX gave the best colorings in staf83 and lsef91 instances while others were one color behind it. Yet, the difference between average colorings and standard deviation is not statistically significant for almost all instances.

We have also compared the best colorings after 10000 generations with some of the results from the literature (Carter et al. [5], Caramia et al. [4] and Merlot et al. [20]). Like DIMACS instances, the performance of the graphs with vertices above 500 suffered due to the limit on the maximum number of generations. For these instances, our crossovers gave similar results in terms of best grouping obtained. Generally they obtained colorings equal or one color behind the colorings of Carter et al. and Caramia et al., and better than those of Merlot et al.

## 5 Conclusion

In this paper, we have investigated the performance of LLE on well known grouping problems, exam timetabling and graph coloring. Several crossover operators that can be used with LLE are represented. The results obtained are promising since LIMX and LIFX perform approximately similar to the two variants of GPX, which is an integral part of the most successful graph coloring algorithm [10]. Also our crossover operators gave satisfactory results for instances in Carter's and DIMACS benchmark suites. In the future, the stochasticity of crossovers which are currently deterministic will be enhanced. LLE will be used on other grouping problems together with the crossover operators aforementioned and their stochastic versions. The multi-objective LLE framework will be used for timetabling problems with additional constraints.

*Acknowledgments.* This research is funded by TUBITAK (The Scientific and Technological Research Council of Turkey) under grant number 105E027.

## References

1. Avanthay, C., Hertz, A., Zufferey, N.: Variable neighborhood search for graph coloring. *European Journal of Operational Research* 151, 379–388 (2003)
2. Brelaz, D.: New methods to color vertices of a graph. *Communications of the ACM* 22, 251–256 (1979)
3. Burke, E.K., Newall, J., Weare, R.F.: A memetic algorithm for university exam timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996)
4. Caramia, M., Dell'Olmo, P., Italiano, G.F.: New algorithms for examination timetabling. In: Näher, S., Wagner, D. (eds.) *WAE 2000*. LNCS, vol. 1982, pp. 230–241. Springer, Heidelberg (2001)
5. Carter, M.W., Laporte, G., Lee, S.T.: Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society* 47, 373–383 (1996)
6. Du, J., Korkmaz, E., Alhadjj, R., Barker, K.: Novel clustering approach that employs genetic algorithm with new representation scheme and multiple objectives. In: Kambayashi, Y., Mohania, M.K., Wöß, W. (eds.) *DaWaK 2004*. LNCS, vol. 3181, pp. 219–233. Springer, Heidelberg (2004)
7. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing* 5, 691–703 (1976)

8. Falkenauer, E.: Genetic Algorithms and Grouping Problems. Wiley, New York (1998)
9. Fleurent, C., Ferland, J.A.: Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research* 63, 437–461 (1996)
10. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization* 3, 379–397 (1999)
11. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979)
12. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA (1989)
13. Holland, J.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (1975)
14. Hertz, A., De Werra, D.: Using tabu search techniques for graph coloring. *Computing* 39, 345–351 (1987)
15. Horn, J., Nafpliotis, N., Goldberg, D.E.: A niched Pareto genetic algorithm for multiobjective optimization. In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, vol. 1, pp. 82–87. IEEE, Piscataway, NJ (1994)
16. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation: Part II, graph coloring and number partitioning. *Operations Research* 39, 378–406 (1991)
17. Johnson, D.S., Trick, M.A.: *Cliques, Coloring and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society, Providence, RI (1996)
18. Kirovski, D., Potkonjak, M.: Efficient coloring of a large spectrum of graphs. In: *35th Design Automation Conference Proceedings*, pp. 427–432 (1998)
19. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards* 84, 79–100 (1979)
20. Merlot, L.T.G., Boland, N., Hughes, B.D., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. In: *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, Gent*, pp. 348–371 (August 2002)
21. Ozcan, E., Ersoy, E.: Final exam scheduler – FES. In: *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1356–1363 (2005)
22. Paquete, L.F., Fonseca, C.M.: A study of examination timetabling with multiobjective evolutionary algorithms. In: *Proceedings of the 4th Metaheuristics International Conference, MIC, Porto*, pp. 149–154 (2001)
23. Radcliffe, N.J.: Formal analysis and random respectful recombination. In: *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 222–229 (1991)
24. Ramani, A., Aloul, F.A., Markov, I., Sakallah, K.A.: Breaking instance-independent symmetries in exact graph coloring. In: *Design Automation and Test Conference in Europe*, pp. 324–329 (2004)
25. Terashima-Marín, H., Ross, P., Valenzuela-Rendón, M.: Clique-based crossover for solving the timetabling problem with gas. In: *Proceedings of the Congress on Evolutionary Computation*, pp. 1200–1206 (1999)
26. Wong, T., Cote, P., Gely, P.: Final exam timetabling: a practical approach. In: *Canadian Conference on Electrical and Computer Engineering, Winnipeg*, vol. 2, pp. 726–731 (2002)

# Ant Algorithms for the Exam Timetabling Problem

Michael Eley

Aschaffenburg University of Applied Science,  
Logistics Laboratory (LLAb), Aschaffenburg, Germany  
`michael.eley@fh-aschaffenburg.de`

**Abstract.** Scheduling exams at universities can be formulated as a combinatorial optimization problem. Basically one has to schedule a certain number of exams in a given number of time periods so that a pre-determined objective function is minimized. In particular, the objective function penalizes schedules where students have to write exams in consecutive periods or even in the same period. Ant colony approaches have been demonstrated to be a powerful solution approach for various combinatorial optimization problems. This paper presents two ant colony approaches for the exam timetabling problem, a Max–Min and an ANTCOL approach. Using the Toronto benchmark test cases from the literature, both algorithms are compared to other timetabling heuristics. Finally, the Max–Min and ANTCOL algorithms are compared using the same set of test cases. In spite of some shortcomings, the ANTCOL approach turned out to be a worthwhile algorithm, among the best currently in use for examination timetabling.

## 1 Introduction

The exam timetabling problem faces the problem of scheduling exams within a limited number of available periods. Setting up a conflict-free timetable is not a trivial task due to limited resources like periods, examination rooms and teacher availability. The main objective is to balance out student workload and to distribute the exams evenly within the planning horizon. In particular, it should be avoided that a student has to write two exams in the same period. Such situations will be referred to as conflicts of order 0 in the sequel. Additionally, as few students as possible have to attend  $r$  exams within  $y$  consecutive periods. Such conflicts can either be totally forbidden by constraints or penalized in the objective function. For example, Carter et al. proposed in [14] a cost function that imposes penalties  $P_\omega$  for a conflict of order  $\omega$ , i.e. whenever one student has to write two exams scheduled within  $\omega + 1$  consecutive periods. In the literature  $\omega$  normally runs from 1 to 5 with  $P_1 = 16, P_2 = 8, P_3 = 4, P_4 = 2, P_5 = 1$ .

Solving practical exam timetabling problems requires that additional constraints have to be considered, e.g. some exams have to be written before other exams or some exams cannot be written within specific periods. References [8] and [13] give comprehensive lists of possible hard and soft constraints.

The exam timetabling problem can be formulated as a graph coloring problem. Each node represents one exam. Undirected arcs connect two nodes if at least one student is enrolled in both corresponding exams. Weights on the arcs represent the number of student enrolled in both exams. The objective is to find a coloring where no adjacent nodes are marked with the same color or to minimize the weighted sum of the arcs that connect two nodes marked with the same color. The exam timetabling problem is a generalization of the graph coloring problem, as in the objective function also conflicts of higher orders are penalized.

A large number of papers presenting heuristic solution approaches to the exam timetabling problem have been published in recent years. Most of the approaches on exam timetabling are modified heuristics derived from graph coloring approaches or use local search methods. Additionally, hyper-heuristics, i.e. heuristics that choose heuristics, have been applied to the exam timetabling problem.

Carter et al. applied in [14] some well known graph coloring heuristics, i.e. saturation degree, largest degree, largest weighted degree, largest enrolment and color degree, which they combined with backtracking. These graph coloring heuristics have been integrated into various other approaches. Asmuni et al. [2] used fuzzy functions to find exams that are difficult to schedule and those should be scheduled early when using graph coloring heuristics.

Di Gaspero and Schearf [20] tested different variants of tabu search based techniques whose neighborhoods concerned those which contributed to the violations of hard or soft constraints. Di Gaspero [19] improved the approach by employing multiple neighborhoods. The first one considers only exams that contribute to the objective function and changes the period of a single exam. The second neighborhood exchanges the periods of two groups of exams at once. White and Xie [39] developed a tabu search approach. This approach was extended in [40] by employing long-term memory. Paquete and Stuetzle [30] developed a tabu search methodology for exam timetabling where ordered priorities were given for the constraints. The length of the tabu list was adaptively set by considering the number of violations in the solutions.

Merlot et al. [27] and Burke et al. [6] developed variants of simulated annealing approaches. While the first paper also uses simulated annealing in combination with constraint programming to generate the initial solution, the latter presents a great deluge algorithm. This approach was further studied in [10] and in [5]. Cote et al. [18] investigated a bi-objective evolutionary algorithm with the objectives of minimizing timetable length and spacing out conflicting exams.

As well as evolutionary algorithms, simulated annealing and tabu search, other local search techniques have been tested to solve exam timetabling problems. Abdullah et al. [1] developed a large neighborhood search based on the methodology of improvement graph construction. Ayob et al. [3] as well as Burke et al. [7] investigated variants of variable neighborhood search. The results of the latter approach were further improved by using a standard genetic algorithm to

intelligently select subset of neighborhoods. Caramia et al. [12] developed a local search method where a greedy scheduler assigned exams into the least possible number of periods, and a penalty decreaser improved the timetable without increasing the number of periods. Finally, Casey and Thompson [15] investigated a greedy randomized adaptive search procedures approach.

The solution quality of many meta-heuristics strongly depends on how well several parameters are chosen. This problem of parameter adjustment led a number of researchers to develop new technologies aimed at operating at a higher level of generality. Kendall and Hussin [24] investigated a tabu search based hyper-heuristic where both moving strategies and constructive graph heuristics were employed as low level heuristics. Yang and Petrovic [41] employed case-based reasoning to choose graph heuristics to construct initial solutions for the Great Deluge algorithm. Burke et al. [9] investigated using tabu search to search permutations of graph heuristics to construct solutions for timetabling problems.

Comprehensive surveys on the literature on exam timetabling problems can be found in [13,37]. In particular, the latter paper lists more than 150 references from journal articles and books published since the mid-1990s.

The aim of this paper is twofold. Originally, this research was motivated by the need for a software tool to practically solving an exam timetabling problem. As ant colony approaches have been demonstrated to be a powerful tool for various combinatorial optimization problems (see the survey in [21]), it is apparent that one can adapt this solution approach to the exam timetabling problem. In the literature different variants of ant colony approaches have been suggested. A comparison of some of these strategies with respect to their suitability for the exam timetabling problem will be made.

This paper is organized as follows. In Section 2 a detailed problem formulation will be presented. Section 3 will give an introduction to ant colony systems. The following sections will present a solution approach and test results for some benchmark problems that are taken from the literature. Finally, Section 6 summarizes the results and discusses suggestions for future work.

## 2 Problem Formulation

Before stating the problem formally, some notation will be introduced.

$R$	index set of rooms
$I$	index set of exams
$T$	index set of periods
$\Omega$	index set of order of conflicts
$K_{rt}$	capacity of room $r$ in period $t$
$c_{ij}$	number of students enrolled in exam $i$ as well as in exam $j$
$E$	total number of students
$E_i$	number of students enrolled in exam $i$
$P_\omega$	penalty imposed if one student has to write two exams within $\omega + 1$ periods

$y_{it}$  binary decision variable equal to 1 if exam  $i$  is scheduled in period  $t$  and 0 otherwise

$p_{irt}$  decision variable indicating the number of students of exam  $i$  assigned to room  $r$  in period  $t$ .

Using this notation, the exam timetabling problem can be formulated as follows:

$$\min \frac{1}{E} \sum_{\omega \in \Omega} \sum_{i,j \in I, i \neq j} \sum_{t \in T, t > \omega} P_{\omega} c_{ij} y_{it} y_{j(t-\omega)} \tag{1}$$

s.t.

$$\sum_{t \in T} y_{it} = 1 \quad \forall i \in I \tag{2}$$

$$p_{irt} \leq y_{it} K_{rt} \quad \forall i \in I, \forall r \in R, \forall t \in T \tag{3}$$

$$\sum_{r \in R} \sum_{t \in T} p_{irt} = E_i \quad \forall i \in I \tag{4}$$

$$\sum_{i \in I} p_{irt} \leq K_{rt} \quad \forall r \in R, \forall t \in T \tag{5}$$

$$\sum_{t \in T} c_{ij} y_{it} y_{jt} = 0 \quad \forall i, j \in I, i \neq j \tag{6}$$

$$y_{it} \in \{0, 1\} \quad \forall i \in I, \forall t \in T \tag{7}$$

$$p_{irt} \in \mathbb{N}_0 \quad \forall i \in I, \forall r \in R, \forall t \in T. \tag{8}$$

The objective function (1) balances out students' workload by minimizing the weighted sum of all conflicts which is divided by the number of students. Constraint (2) states that each exam is assigned to exactly one period. If an exam is not assigned within a period, then no seats should be reserved for that period in any room. This is imposed by constraint (3). Constraints (4) and (5) ensure that the number of seats reserved for an exam will be equal to the number of students who are enrolled in that exam and that the room capacities are not exceeded. Finally, constraint (6) avoids conflicts of order 0, i.e. that a student has to write two exams in the same period.

The exam timetabling problem is a generalization of the graph coloring problem, which is known to be NP-hard [23]. Therefore, to solve large real-world problems within a reasonable amount of time, heuristics are used. In the following sections a solution approach will be presented. But instead of considering capacity constraints for the single rooms, only the total capacity of all available exam rooms within a period will be considered. In the IP formulation stated above this can be accomplished by replacing the set of rooms by an artificial single room.

### 3 Ant Algorithms

Ant colony optimization algorithms represent special solution approaches for combinatorial optimization problems derived from the field of swarm intelligence. They were first introduced by Colormi et al. in the early 1990s [16]. See [21] for an in-depth introduction to ant systems.

Ant algorithms were inspired by the observation of how real ant colonies find shortest paths between food sources and their nest. This observation was first implemented in algorithms for solving the traveling salesperson problem (TSP). This type of ant colony optimization algorithm is known in the literature as ant system (AS) algorithms. The basic principle of AS algorithms will be described briefly by means of the TSP. This solution approach to the TSP will then be adopted to solving the exam timetabling problem in the next section.

The solution approach consists of  $n$  cycles. In each of these cycles first each of the  $m$  ants constructs a feasible solution. In AS each ant builds a complete tour that visits all nodes. Obviously, this solution neither has to be optimal nor must it be even close to the (unknown) optimal value. Improved solutions can be obtained if knowledge gathered by other ants in the past on how high quality solutions can be obtained, is incorporated into the ants' decision. Assume that an ant is located in a node  $i$ . To choose the next node  $j$  that has not yet been visited by that ant one may apply one of the following two randomized strategies:

**Strategy I:** *Constructive heuristic.* Apply one priority rule like randomized nearest neighbor. Decision values for all nodes  $j$  are determined by the inverse of the distance from node  $i$  to that node  $j$ . The next node the ant moves to is then randomly chosen according to the probabilities determined by those decision values. Consequently, if node  $j_1$  is closer to  $i$  than node  $j_2$  it is more likely to choose node  $j_1$ . The decision values of the constructive heuristic will be referred to as  $\eta_{ij}$ .

**Strategy II:** *Pheromone trails.* This strategy is mainly inspired by the way real ants find shortest paths. While commuting between two places on different possible paths ants deposit a chemical substance called pheromone. The shorter the path is the more often the ant will use this path within a limited period of time and, consequently, the larger the amount of pheromone will be on that path. Thus, whenever an ant has to choose between different available paths it will prefer the one with higher amount of pheromone.

To adapt these observations to the TSP, the amount of pheromone is stored in a matrix  $\tau$ . Each cell  $\tau_{ij}$  of the matrix is associated with an arc  $(ij)$  and the matrix is initialized with 0 for all cells. After an ant has completed a tour, the values of the cells that belong to the arcs the ant has chosen are updated by the inverse of the obtained objective function value, i.e. the length of the tour. The amount of pheromone trail  $\tau_{ij}$  associated to arc  $(i, j)$  is intended to represent the learned desirability of choosing node  $j$  when in node  $i$ . Consequently, arcs that belong to good solutions receive a high amount of pheromone.



AS algorithms combine these two strategies. The probability that an ant  $\nu$  located in node  $i$  chooses the next node  $j$  is determined by the following formula:

$$p_{ij}^{\nu} = \begin{cases} \frac{(\tau_{ij})^{\alpha} (\eta_{ij})^{\beta}}{\sum_{k \in N_i^{\nu}} (\tau_{ik})^{\alpha} (\eta_{ik})^{\beta}} & \text{if } j \in N_i^{\nu} \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

$\alpha$  and  $\beta$  are a given weighting factors and  $N_i^{\nu}$  is the set of nodes that have not yet been visited by ant  $\nu$  currently located in node  $i$  (see [21]).

Excepting the TSP, AS algorithms have been implemented for various combinatorial optimization problems, such as the quadratic assignment problem or the sequential ordering problem. Different variants of AS algorithms have been suggested in the literature, for example ant colony systems (ACS) and Max–Min ant systems (MMAS). Both produced much better results than AS (see [21]). In particular, MMAS, which was first proposed by Stuetzle and Hoos [33], generated significantly better solutions for the TSP than AS. Socha et al. [32] compared the MMAS variant with ACS and found that MMAS outperformed the ACS approach for the considered timetabling problem.

The main modifications of MMAS are related to the way that the matrix  $\tau$  is initialized and how the pheromone values are updated. Additionally, MMAS uses local search to improve the solutions found by the ants. Details are discussed in the next section.

Ant colony algorithms have been used recently to solve different types of scheduling problems, see for example Blum [4]. Socha et al. [31,32] developed algorithms for the timetabling problem for university classes, which is slightly different from the exam timetabling problem considered here. Costa and Hertz [17] used an ant colony approach to solve assignment type problems, in particular graph coloring problems. They pursued the objective of minimizing the number of periods required for a clash-free timetable. Recently, Vesel and Zerovnik [38] as well as Dowsland and Thompson modified and improved [22] this graph coloring algorithm. Whilst the first article puts the focus on graph coloring problems, the latter improves the approach of Costa and Hertz [17] with respect to the examination scheduling problem, by introducing new initialization methods, trial calculations and fitness functions.

Finally, Naji Azimi [28] implemented an ACS algorithm and compared it to simulated annealing, a genetic algorithm and tabu search. These results with randomly generated test problems indicated that the ant-based approach outperformed the other approaches. Additionally, the author tested some hybrid versions where ACS was combined with tabu search. These hybrid versions outperformed all meta-heuristics [29].

## 4 An Ant Algorithm for the Exam Timetabling Problem

### 4.1 General Modifications for the Exam Timetabling Problem

In this section a description will be given on how the AS algorithm must be modified in order to solve exam timetabling problems. The solution approach

consists of  $n$  cycles. Firstly, in each of these cycles each of the  $m$  ants constructs a feasible solution therefore using the constructive heuristic and the pheromone trails. These exam schedules are then evaluated according to the given objective function and the experience accumulated during the cycle is used to update the pheromone trails.

Depending on the choice of a constructive heuristic and the way the pheromone values are used, there are different ways in which this basic solution approach can be adapted to the exam timetabling problem.

- In Socha et al. [32] a pre-ordered list of events is given. Each ant chooses the color for a given node probabilistically similar to the formula (9). The pheromone trail  $\tau_{Ij}$  contains information on how good the solution was, when node  $i$  was colored by color  $t$ . The constructive heuristic employed in their approach is not described.
- Socha et al. [31] suggested two variants of how the basic solution approach presented in the last section can be adapted to the (course) timetabling problem:
  - A list of periods is given. Starting with the first period in the list, each ant assigns courses to this period.
  - A list of courses is given and the ants chooses a period for each course of the list while starting with the first course from the list. The period is then chosen probabilistically according to a formula similar to (9).

The authors preferred the second variant as it seemed to be a more natural approach, in particular when the number of events is larger than the number of periods. Additionally, the authors tested two different representations for the pheromone values in the matrix  $\tau$ :

- Direct representation. A cell  $\tau_{it}$  of the matrix represents the amount of pheromone if course  $i$  is assigned to a period  $t$ .
- Indirect representation. A cell  $\tau_{ij}$  of the matrix indicates if courses  $i$  and  $j$  should be assigned to the same period.

Experiments indicted that the indirect representation produced better results, in particular when an additional hill climber was used.

- In the ACS approach of Naji Azimi [29] each ant follows a list of exams and chooses a period as in [31]. The period is chosen randomly with probabilities depending on the pheromone matrix and heuristic information. The paper gives no information on how a time list of exams is sorted.
- At each stage of the construction process in the AS approach of Costa and Hertz [17] called ANTCOL the ant chooses first a node  $i$  according to a probability distribution equivalent to (9) and then a feasible color. Experiments showed that also choosing the color probabilistically did not improve the solutions. As in [31] an indirect representation was employed. The matrix  $\tau$  provides information on the objective function value, i.e. the number of colors required to color the graph, which was obtained when nodes  $i$  and  $j$  are colored with the same color.

In contrast to elite strategies where only the ant that found the best tour from the beginning of the trial deposits pheromone, all ants deposit pheromone on the paths they have chosen. According to [21] this strategy is called ant cycle strategy.

Different priority rules were tested as constructive heuristic. Among those chosen in each step, the node with the highest degree of saturation, i.e. the number of different colors already assigned to adjacent nodes, achieved the best results with respect to solution quality and computation times.

For the exam timetabling problem the way that the information in matrix  $\tau$  is used in both approaches seems not to be meaningful. Due to the conflicts of higher orders the quality of a solution does not depend on how a pair of exams is scheduled, nor does it depend on the specific period an exam is assigned to. For example, assigning two exams  $i$  and  $j$  with  $c_{ij} = 0$  to the same period can either result in a high or in a low objective function value as the quality of the solution strongly depends on when the remaining exams are scheduled. In the following a two-step approach was implemented.

**Step I:** Determine the sequence according to the exams are scheduled. As for the TSP we assume that an ant located in a node, corresponding to an exam, has to visit all other nodes, i.e. it has to construct a complete tour. The sequence according to which this ant constructs the tour corresponds to the sequence in which the exams are scheduled. Thus,  $\tau_{ij}$  indicates how advantageous it is to choose exam  $j$  as the  $i$ th exam in the sequence.

The rationale behind this idea is that the ants should learn which exams should be scheduled early, in order to avoid high penalties in the objective function in later iterations. Asmuni et al. [2] as well as Merlot et al. [27] pursued similar concepts. Whilst in the first paper fuzzy functions were employed when ordering the exams on how difficult they were, the latter ordered the exams by the size of their domains (available periods) and scheduled them into the earliest period one by one.

**Step II:** Find the most suitable period for an exam which should be scheduled. As recommended by Costa and Hertz [17] the period is not chosen probabilistically. Instead, all admissible periods are evaluated according to the given penalty function and the exam is scheduled in the period that achieved the best evaluation. Thus, as recommended by Dowsland and Thompson [22] the construction approach is not restricted to filling one period before starting another.

If exam  $j$  has a high  $\tau_{ij}$  value for a small value of  $i$ , this exam should be scheduled early in the sequence. Assume that this exam is by chance not chosen as number  $i$  in the sequence. Then values  $\tau_{k,j}$  for  $k = i+1, i+2, \dots$  should also be high in order to make sure that the ant will choose  $j$  soon. Therefore, Merkle and Middendorf modified (9) by the following so-called pheromone summation rule [26]:

$$p_{ij}^\nu = \begin{cases} \frac{(\sum_{h=1}^i [\tau_{hj}])^\alpha (\eta_{ij})^\beta}{\sum_{k \in N_i^\nu} (\sum_{h=1}^i [\tau_{hk}])^\alpha (\eta_{ik})^\beta} & \text{if } j \in N_i^\nu \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Pheromone values  $\tau_{ij}$  along the ants' paths are updated by the inverse of the objective function value. For the heuristic value  $\eta_{ij}$  the following simple priority rule for graph coloring was implemented. The exam with the smallest number of available periods is selected. A period would not be available for an exam if it caused a conflict of order 0 with another exam that has already been scheduled. This priority rule corresponds to the saturation degree rule (SD) which was tested in [14]. The value  $\eta_{ij}$  is chosen to be the inverse of the saturation degree.

## 4.2 MMAS Specifications

MMAS approaches mainly differ from AS algorithms in the way they use the existing information (see [33]):

- Pheromone trails are only updated by the ant that generated the best solution in a cycle. The corresponding values  $\tau_{ij}$  are updated by  $\rho\tau_{ij} + 1/f^{best}$  where  $f^{best}$  is equal to the best objective function value found so far. For all other arcs  $(i, j)$  that are not chosen by the best ant  $\tau_{ij}$  is updated by  $(1 - \rho)\tau_{ij}$ , where  $\rho \in [0, 1]$  represents the pheromone evaporation factor, i.e. the percentage of pheromone that decays within a cycle.
- Pheromone trail values are restricted to the interval  $[\tau_{min}, \tau_{max}]$ , i.e. whenever after a trail update  $\tau_{ij} < \tau_{min}$  or  $\tau_{ij} > \tau_{max}$  then  $\tau_{ij}$  is set to  $\tau_{min}$  or  $\tau_{max}$ , respectively. The rationale behind this is that if the differences between some pheromone values were too large, all ants would almost always generate the same solutions. Thus, stagnation is avoided.
- Pheromone trails are initialized to their maximum values  $\tau_{max}$ . This type of pheromone trail initialization increases the exploration of solutions during the first cycle.

The solution quality of ant colony algorithms can be considerably improved when it is combined with additional local search. In hybrid MMAS only the best solution within one cycle is improved by local search. For the exam timetabling problem a hill climber procedure has been implemented. Within an iteration of the hill climber two sub-procedures are carried out in succession. The hill climber is stopped if no improvement can be found within an iteration.

Within the first sub-procedure of the hill climber for all exams the most suitable period is examined. Beginning with the exam that causes the biggest contribution to the objective function value, all feasible periods are checked and the exam is assigned to its best period. The first sub-procedure is stopped if all exams have been checked without finding an improvement. Otherwise the contributions to the objective function value are recalculated and the process is repeated.

The second sub-procedure tries to decrease the objective function value by swapping all exams within two periods, i.e. all exams assigned to period  $t'$  are moved to period  $t''$  and the exams of that period are moved to period  $t'$ . Therefore all pairs of periods are examined and the first exchange that leads to an improvement is carried out. Again, the process is repeated as long as the objective function value is decreased.

Finally, the use of a so-called candidate can reduce required computational times as well as improve solution quality at the same time (see [21]). Such a list provides additional local heuristic information as it contains preferred nodes to be visited from a given node. Instead of scanning all other exams, only the exams in the candidate list are examined, and only when all exams in this list have been scheduled, are the remaining exams considered.

## 5 Computational Experiments

The proposed Max–Min algorithm was implemented in Borland Delphi 7.0. It will be referred to as MMAS-ET in the sequel. Test runs were carried out on a computer with 3.2 GHz clock under Windows XP.

### 5.1 Test Cases

To benchmark algorithms test cases of thirteen practical examination problems can be found on the site of Carter (see [34]). Table 1 summarizes some characteristics of these problems. These test problems are also known in the literature as the Toronto data set version I (see [37]).

To make a comparison meaningful all algorithms must use the same objective function. Therefore, Carter proposed weighting conflicts according to the following penalty function:  $P_1 = 16$ ,  $P_2 = 8$ ,  $P_3 = 4$ ,  $P_4 = 2$ ,  $P_5 = 1$ , where  $P_\omega$  is the penalty for the constraint violation of order  $\omega$ . The cost of each conflict is multiplied by the number of students involved in both exams. The objective function value represents the costs per student. As the proposed MMAS-ET algorithm does not guarantee that no conflicts of order 0 occur, additionally, the penalty  $P_0$  was imposed and set to 10000.

### 5.2 Adjustment of the Parameters

The required parameters were specified as follows. The number of cycles was set to 50. Within each cycle 50 ants were employed to construct solutions. The candidate list contained the 20% of exams with the lowest number of available periods. Several test runs were carried out in order to determine the required parameters appropriately:

- The evaporation rate  $\rho$  was set to 0.3. As in [33] it turned out that this parameter is quite robust, i.e. the parameter  $\rho$  does not clearly influence the performance.
- For the restrictions of the pheromone interval values to strategies were tested. Setting  $\tau_{max} = 1/\rho$  obtained slightly better results than in the case of variable  $\tau_{max}$  and  $\tau_{min}$  as proposed in [33].
- Different values for the weighting factors  $\alpha$  and  $\beta$  were tested. It turned out that the approach performed best when  $\alpha$  was set to one and  $\beta$  was chosen high. Best results were obtained for  $\beta$  equal to 10. But the difference was on average less than one percent when  $\beta$  was bigger than eight. A high  $\beta$  forces

**Table 1.** Toronto data set version I from Carter et al. [14,34,35]

Test case	Number of exams	Number of students	Number of student exams	Problem density	Number of periods
car92I	543	18419	55522	13.8%	32
car91I	682	16925	56877	12.8%	35
ear83I	190	1125	8109	26.7%	24
hec92I	81	2823	10632	42.0%	18
kfu93I	461	5349	25113	5.6%	20
lse91I	381	2726	10918	6.3%	18
rye92I	486	11483	45051	7.5%	23
sta83I	139	611	5751	14.4%	13
tre92I	261	4360	14901	5.8%	23
uta92I	622	21267	58979	12.6%	35
ute92I	184	2750	11793	8.5%	10
yor83I	181	941	6034	28.9%	21
pur93I	2419	30032	120681	2.9 %	43

that exams which can be scheduled, due to zero-order conflicts, only in a few remaining periods are scheduled first, as they are given a much higher probability in (9). Remember that  $\eta_{ij}$  is the inverse of the saturation degree as explained in Section 4.1. Thus, a high  $\beta$  value has the same effect like a candidate list. This could be a reason why the use of the candidate list did not improve the solutions. For small values of  $\beta$ , i.e. values lower than 5, solutions with zero-order conflicts could not always be avoided.

- As the approach is non-deterministic each test case was solved 20 times.

After determining the parameters in such a way, it turned out that less than 2% of the solutions were generated more than once. Thus, stagnation, that is caused by the fact that many ants generate almost the same solutions, could not be observed.

### 5.3 Test Results for the MMAS-ET Approach

Table 2 displays the results for different approaches. The solutions are also available on the Internet [36]. For each approach the minimal objective function value and the average result after 20 test runs are given in Table 2. Results of the proposed MMAS-ET approach are given in the second column.

In order to find out how much the hill climber contributes to the solution, the MMAS-ET approach was also tested without making use of the hill climber. Comparing the results in the second and in the third column it is obvious that the hill climber considerably improves the solutions.

Increasing the number of ants and the number of cycles to 100 in the MMAS-ET approach did not result in achieving better solutions. Neither the average value of all 20 iterations was improved nor were better solutions found during the 20 iterations.

**Table 2.** Results (objective function values) for two variants of the MMAS-ET approach for 20 test runs

Test case	MMAS-ET		MMAS-ET without hill climber	
	Best	Avg.	Best	Avg.
car92I	4.7	4.8	7.7	7.9
car91I	5.7	5.8	9.4	9.6
ear83I	36.8	38.3	50.4	53.8
hec92I	11.2	11.4	15.0	15.5
kfu93I	15.0	15.4	24.0	24.8
lse91I	12.2	12.6	18.9	19.6
rye93I	10.1	10.3	17.9	18.5
sta83I	157.4	157.6	162.1	163.6
tre92I	8.9	9.2	12.2	12.7
uta92I	3.8	3.8	6.0	6.2
ute92I	27.7	28.4	32.9	34.4
yor83I	39.3	40.2	50.2	51.4
pur93I	5.5	5.6	12.2	12.5

#### 5.4 Comparison with the Approach of Costa and Hertz

The results of MMAS-ET were compared with a modified version of the ANTCOL algorithm of Costa and Hertz [17], which was originally developed for solving graph coloring problems. This approach will be called ANTCOL-ET in the sequel. Within that approach the ANT\_DSATUR(1) procedure was used as a constructive method as described in [17]. Also the objective function was modified in order to consider conflicts of higher order. Test runs were carried out to adjust the parameters appropriately. The parameter  $\alpha$  was set to 1,  $\beta$  to 30.  $\rho$  was set equal to 0.3. Again, each test case was solved 20 times.

Table 3 shows the results for the 13 test cases and compares them with the MMAS-ET approach. Surprisingly, the simple AS-like approach ANTCOL-ET outperformed the MMAS-ET for some test cases. In particular, this result is contrary to other results presented in the literature where MMAS algorithms obtained better results for various combinatorial optimization problems by avoiding stagnation (see [21,33]).

Thus, ANTCOL-ET was modified by implementing additionally the hill climber already incorporated in the MMAS-ET approach. This modified version of the Costa and Hertz approach provided, on average, better solutions than the MMAS-ET approach. Obviously, the combination of an indirect representation plus a hill climber is capable of generating good solutions. Socha et al. [31] made a similar observation for the course timetabling problem.

Computing times for the MMAS-ET approach lay in the range of 10 seconds for the smallest test cases, i.e. hec-s-92, to 2.5 hours for the pur-s-93 problem. Compared to the MMAS-ET approach the computing time of the ANTCOL-ET combined with the hill climber was on the average 80% higher. Thus, one

**Table 3.** Comparison of the objective function values of different ant colony approaches

Stopping criterion		MMAS-ET	ANTCOL-ET	ANTCOL-ET	ANTCOL-ET
		2500 solutions	2500 solutions	without hill climber	with hill climber
				2500 solutions	Same running time as MMAS-ET
car92I	best	4.7	4.5	4.3	4.3
	avg.	4.8	4.6	4.4	4.4
car91I	best	5.7	5.3	5.2	5.2
	avg.	5.8	5.4	5.2	5.3
ear83I	best	36.8	40.3	36.8	38.1
	avg.	38.3	41.4	38.3	38.5
hec92I	best	11.2	12.2	11.1	11.2
	avg.	11.4	12.6	11.4	11.4
kfu93I	best	15.0	15.4	14.5	14.6
	avg.	15.4	15.8	14.9	14.9
lse91I	best	12.2	11.9	11.3	11.4
	avg.	12.6	12.2	11.7	11.7
rye93I	best	10.1	10.2	9.8	9.8
	avg.	10.3	10.7	10.0	10.0
sta83I	best	157.4	158.2	157.3	157.3
	avg.	157.6	159.3	157.5	157.5
tre92I	best	8.9	8.8	8.6	8.6
	avg.	9.2	9.0	8.7	8.7
uta92I	best	3.8	3.6	3.5	3.5
	avg.	3.8	3.7	3.5	3.5
ute92I	best	27.7	28.9	26.4	26.7
	avg.	28.4	29.4	27.0	27.5
yor83I	best	39.3	42.2	39.4	40.1
	avg.	40.2	43.7	40.4	40.7
pur93I	best	5.5	4.8	4.6	4.6
	avg.	5.6	4.9	4.6	4.6

can conclude that ANTCOL-ET takes more time but gets a better solution quality than MMAS-ET. Note that the same stopping criterion was used for both algorithms, namely 2500 solutions.

Additionally, test runs for the ANTCOL-ET approach were carried out where the running time was limited to the time that the MMAS-ET approach required to generate 2500 solutions. The results are displayed in the last column of Table 3. Thus, comparing both algorithms with the same running time the ANTCOL-ET approach outperformed the MMAS-ET approach in ten out of the thirteen test cases.

Finally, a second variant of the ANTCOL-ET approach was implemented. As in the approach of Costa and Hertz an indirect representation was employed in this variant. But instead of using the AS framework, the pheromone trails were initialized and updated as in the MMAS approach described in Section 4.2. Also



**Table 4.** Best (b.) and average (a.) objective function value for the Toronto data set version I (best solutions printed in bold)

Test case	Cal96	Cal01	DGS01	DG02	PS02	BN03	Mal03	Wal04	BN04	Bal04	Aal05
	<b>[14]</b>	<b>[12]</b>	<b>[20]</b>	<b>[19]</b>	<b>[30]</b>	<b>[10]</b>	<b>[27]</b>	<b>[40]</b>	<b>[11]</b>	<b>[6]</b>	<b>[2]</b>
car91I b.	7.1	6.6	6.2	5.7	-	4.6	5.1	5.7	5.0	4.8	5.3
a.	7.1	6.6	6.5	5.8	-	4.7	5.2	5.8	-	6.1	-
car92I b.	6.2	6.0	5.2	-	-	4.0	4.3	4.6	4.3	4.2	4.6
a.	6.2	6.0	5.6	-	-	4.1	4.4	4.7	-	4.3	-
ear83I b.	36.4	<b>29.3</b>	45.7	39.4	40.5	36.1	35.1	45.8	36.2	35.4	37.0
a.	36.4	<b>29.3</b>	46.7	43.9	45.8	37.1	35.4	46.4	-	36.7	-
hec92I b.	10.8	<b>9.2</b>	12.4	10.9	11.7	11.3	10.6	12.9	11.6	10.8	11.8
a.	10.8	<b>9.2</b>	12.6	11.4	12.4	11.5	10.7	13.4	-	11.5	-
kfu93I b.	14.0	13.8	18.0	-	16.5	13.7	13.5	17.1	15.0	13.7	15.8
a.	14.0	<b>13.8</b>	19.5	-	18.3	13.9	14.0	17.8	-	14.4	-
lse91I b.	10.5	<b>9.6</b>	15.5	12.6	13.2	10.6	10.5	14.7	11.0	10.4	12.1
a.	10.5	<b>9.6</b>	15.9	13.0	15.5	10.8	11.0	14.8	-	11.0	-
rye92I b.	7.3	<b>6.8</b>	-	-	-	-	8.4	11.6	-	8.9	10.4
a.	7.3	<b>6.8</b>	-	-	-	-	8.7	11.7	-	9.3	-
sta83I b.	161.5	158.2	160.8	157.4	161.2	168.3	157.3	158.0	161.9	159.1	160.4
a.	161.5	158.2	167.0	157.7	168.7	168.7	157.4	158.0	-	159.4	-
tre92I b.	9.6	9.4	10.0	-	9.3	8.2	8.4	8.9	8.4	8.3	8.7
a.	9.6	9.4	10.5	-	10.2	8.4	8.6	9.2	-	8.4	-
uta92I b.	3.5	3.5	4.2	4.1	-	3.2	3.5	4.4	3.4	3.4	3.6
a.	3.5	3.5	4.5	4.3	-	3.2	3.6	4.5	-	3.5	-
ute92I b.	25.8	<b>24.4</b>	27.8	-	28.7	25.5	25.1	29.0	27.4	25.7	27.8
a.	25.8	<b>24.4</b>	31.3	-	30.5	25.8	25.2	29.1	-	26.2	-
yor83I b.	41.7	36.2	41.0	39.7	38.9	36.8	37.4	42.3	40.8	36.7	40.7
a.	41.7	<b>36.2</b>	42.1	40.6	41.7	37.3	37.9	42.5	-	37.2	-
$\geq$ MMAS	5	4	11	6	7	2	0	11	4	1	7
$\geq$ ANTCOL	6	5	11	6	7	2	3	12	6	1	12

Cal96: Carter et al.; Cal01: Caramia et al.; DGS01: Di Gaspero and Schaefer;  
 DG02: Di Gaspero; PS02: Paquete and Stuetzle (Lex-seq approach);  
 BN03: Burke and Newall; Mal03: Merlot et al.; Wal04: White et al.;  
 BN04: Burke and Newall; Bal04: Burke et al.; Aal05: Asmuni et al.

the trail values were restricted and the hill climber was used. This MMAS variant of the approach of Costa and Hertz generated solutions comparable to the MMAS-ET approach. Only for the sta83I test case was the objective function value of 157.3 for the best solution better than the best solution generated by the MMAS-ET approach. And for the yor83I test case both other approaches were outperformed with an objective function value of 39.2. Thus, the test results indicate that, irrespective of the representation, the ANTCOL approach outperforms the MMAS approach.

**Table 5.** Best (b.) and average (a.) objective function value for the Toronto data set version I (best solutions printed in bold)

Test case	Cal05	KH05	YP05	Aal06	Bal06a	Bal06b	ABK	BB06	MMAS	ANTCOL
	<b>[18]</b>	<b>[24]</b>	<b>[41]</b>	<b>[1]</b>	<b>[9]</b>	<b>[7]</b>	<b>[3]</b>	<b>[5]</b>	-ET	-ET
car91I b.	5.4	5.4	4.5	5.2	5.4	4.6	4.5	<b>4.4</b>	5.7	5.2
a.	5.5	-	<b>4.5</b>	-	-	-	-	-	5.8	5.2
car92I b.	4.2	4.7	3.9	4.4	4.5	4.0	4.9	<b>3.7</b>	4.7	4.3
a.	4.3	-	<b>4.0</b>	-	-	-	-	-	4.8	4.4
ear83I b.	34.2	40.2	33.7	34.9	37.9	32.8	36.3	32.8	36.8	36.8
a.	35.6	-	34.9	-	-	-	-	-	38.3	38.3
hec92I b.	10.4	11.9	10.8	10.3	12.3	10.0	11.1	10.2	11.2	11.1
a.	10.5	-	11.4	-	-	-	-	-	11.4	11.4
kfu93I b.	14.3	15.8	13.8	13.5	15.2	<b>13.0</b>	14.7	<b>13.0</b>	15.0	14.5
a.	14.4	-	14.4	-	-	-	-	-	15.4	14.9
lse91I b.	11.3	-	10.4	10.2	11.3	10.0	12.1	9.8	12.2	11.3
a.	11.5	-	10.8	-	-	-	-	-	12.6	11.7
rye92I b.	8.8	-	8.5	8.7	-	-	10.7	-	10.1	9.8
a.	9.1	-	8.8	-	-	-	-	-	10.3	10.0
sta83I b.	<b>157.0</b>	157.4	158.4	159.2	158.2	159.9	157.3	<b>157.0</b>	157.4	157.3
a.	<b>157.1</b>	-	-	-	-	-	-	-	157.6	157.5
tre92I b.	8.6	8.4	7.9	8.4	8.9	7.9	8.9	<b>7.8</b>	8.9	8.6
a.	8.8	-	<b>8.1</b>	-	-	-	-	-	9.2	8.7
uta92I b.	3.5	-	<b>3.1</b>	3.6	3.9	3.2	3.6	<b>3.1</b>	3.8	3.5
a.	3.6	-	<b>3.2</b>	-	-	-	-	-	3.8	3.5
ute92I b.	25.3	27.6	25.4	26.0	28.0	24.8	26.4	24.8	27.7	26.4
a.	25.5	-	26.1	-	-	-	-	-	28.4	27.0
yor83I b.	36.4	-	36.5	36.2	41.4	37.3	39.0	<b>34.8</b>	39.3	39.4
a.	37.6	-	36.9	-	-	-	-	-	40.2	40.4
≥MMAS-ET	0	3	1	1	8	1	3	0		
≥ANTCOL	4	7	1	4	11	1	7	0		

Cal05: Cote et al.; KH05: Kendall and Hissan; YP05: Yang and Petrovic; Aal06: Abdullah et al.; Bal06a: Burke et al.; Bal06b: Burke et al.; ABK06: Ayob et al.; BB06: Burke and Bykov

### 5.5 Comparison with Other Exam Timetabling Approaches

The MMAS-ET as well as the ANTCOL-ET approach were compared with approaches from the literature. These benchmark approaches minimize the objective function of Carter et al. presented in Section 5.1. The approaches have been tested using the Toronto data set version I. Results of the benchmarks are taken from the literature [\[37,40\]](#) and from the internet (see the timetabling database at the University of Melbourne [\[35\]](#)).

Tables [4](#) and [5](#) display the best solution and the average solution achieved. The results can be summarized as follows. Although neither the MMAS-ET nor the ANTCOL-ET approach can improve the best solution for any of the 12 test cases,

their performance is comparable with most of the 19 benchmark algorithms. It is striking that no approach outperforms all other approaches for all test cases. Therefore, the last two lines in Tables 4 and 5 indicate how often the MMAS-ET approach and the ANTCOL-ET approach, respectively, obtained solutions that were not worse than the benchmark approach corresponding to the column in the table.

The ANTCOL-ET approach is capable of finding better solutions for at least some test cases compared to almost all benchmark approaches. For example, the highly competitive Bal04 approach of Burke et al. was outperformed for one test case, namely sta83I. Good results were obtained in particular for the test cases sta83I, car92I, uta92I and car91I. Only the recently published algorithm of Burke and Bykov [5] clearly dominates both ant algorithms for all test cases. But, for this flex-deluge approach, running times between five and ten hours were reported which are considerably higher than for ANTCOL-ET.

There are also some test cases where MMAS-ET outperforms the approaches. For example, MMAS-ET generates better solutions than the Cal01 approach of Caramia et al. (that holds the best results in five out of the twelve test cases) in four out of the twelve test cases, i.e. for the test cases car91I, car92I, sta83I and tre92I. White et al. argued in [40] that these test cases seem to be in a way easier than the other test cases.

## 6 Conclusion

In this paper different strategies for solving exam timetabling problems by ant algorithms have been implemented. In ant colony optimization the search for good solutions is incorporated in the learning process controlled by the definition of the pheromone trail and the constructive heuristic. Different representations for the pheromone values have been tested as well as different strategies for updating the pheromone, i.e. ant systems (AS) and Max–Min ant systems (MMAS).

The most effective ant algorithm turned out to be a modified version of the AS approach of Costa and Hertz [17]. In particular, a hill climber was added to this approach that improves the solutions and has a strong impact on the solution quality. Unlike other combinatorial optimization problems, for example the TSP or the QAP, the exam timetabling problem using the MMAS approach did not outperform the simpler AS strategy.

Of course, it goes without saying that proper adjusting parameters can improve the performance of an algorithm considerably. In particular the values of  $\alpha$  and  $\beta$  have a strong impact on the solution quality.

The implemented algorithms have been compared with the existing literature on the problem. Unfortunately, the experimental analysis shows that the results of our algorithms are not satisfactory on all benchmark instances. Nevertheless, we consider these results quite encouraging, and they provide a good basis for future improvements. One promising direction could be the parallelization of the ANTCOL-ET or the MMAS-ET approach. Recently, Manfrin et al. [25] tested different interconnection topologies for a Max–Min approach to the TSP.

They showed that the parallel models outperformed the equivalent sequential algorithms.

Another self-evident extension of the approach would be to incorporate additional constraints and requirements, like for example scarce room resources or precedence constraints between exams.

## References

1. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja–Orlins large neighbourhood search for examination timetabling. *OR Spectrum* 29, 331–372 (2007)
2. Asmuni, H., Burke, E.K., Garibaldi, J., McCollum, B.: Fuzzy multiple ordering criteria for examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 334–353. Springer, Heidelberg (2005)
3. Ayob, M., Burke, E.K., Kendall, G.: An iterative re-start variable neighbourhood search for the examination timetabling problem. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 336–344 (August 2006)
4. Blum, C.: *Theoretical and Practical Aspects of Ant Colony Optimization*. Akademische Verlagsgesellschaft, Berlin (2004)
5. Burke, E.K., Bykov, Y.: Solving exam timetabling problems with the flex-deluge algorithm. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, pp. 370–372 (August 2006)
6. Burke, E.K., Bykov, Y., Newall, J.P., Petrovic, S.: A time-predefined local search approach to exam timetabling problems. *IIE Transactions* 36, 509–528 (2004)
7. Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. Technical Report NOTTCS-TR-2006-2, University of Nottingham (2006)
8. Burke, E.K., Elliman, D.G., Ford, P.H., Weare, R.F.: Examination timetabling in British universities: A survey. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 76–92. Springer, Heidelberg (1996)
9. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph based hyper-heuristic for exam timetabling problems. *European Journal of Operational Research* 176, 177–192 (2007)
10. Burke, E.K., Newall, J.P.: Enhancing timetable solutions with local search methods. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 195–206. Springer, Heidelberg (2003)
11. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of Operational Research* 129, 107–134 (2004)
12. Caramia, M., Dell’Olmo, P., Italiano, G.F.: New algorithms for examination timetabling. In: Swierstra, S.D. (ed.) *PLILP 1995*. LNCS, vol. 982, pp. 230–241. Springer, Heidelberg (1995)
13. Carter, M.W., Laporte, G.: Recent developments in practical examination timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996)
14. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling algorithmic strategies and applications. *Journal of the Operational Research Society* 47, 373–383 (1996)

15. Casey, S., Thompson, J.: Grasping the examination scheduling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 233–244. Springer, Heidelberg (2003)
16. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Proceedings of the 1st European Conference on Artificial Life, pp. 134–142. Elsevier, Amsterdam (1992)
17. Costa, D., Hertz, A.: Ants can color graphs. *Journal of the Operational Research Society* 48, 295–305 (1997)
18. Cote, P., Wong, T., Sabouri, R.: Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 151–168. Springer, Heidelberg (2005)
19. Di Gaspero, L.: Recolour, shake and kick: A recipe for the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, Springer, Heidelberg (2003)
20. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001)
21. Dorigo, M., Di Caro, G., Gambarella, L.M.: Ant algorithms for discrete optimization. *Artificial Life* 5, 137–172 (1999)
22. Dowsland, K., Thompson, J.: Ant colony optimisation for the examination scheduling problem. *Journal of the Operational Research Society* 56, 426–439 (2005)
23. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York (1979)
24. Kendall, G., Hussin, N.M.: A tabu search hyper-heuristic approach to the examination timetabling problem at the Mara University of Technology. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 199–218. Springer, Heidelberg (2005)
25. Manfrin, M., Birattari, M., Stuetzle, T., Dorigo, M.: Parallel ant colony optimization for the traveling salesman problem. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 224–234. Springer, Heidelberg (2006)
26. Merkle, M., Middendorf, M.: An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: Oates, M.J., Lanzi, P.L., Li, Y., Cagnoni, S., Corne, D.W., Fogarty, T.C., Poli, R., Smith, G.D. (eds.) EvoIASP 2000, EvoWorkshops 2000, EvoFlight 2000, EvoSCONDI 2000, EvoSTIM 2000, EvoTEL 2000, and EvoROB/EvoRobot 2000. LNCS, vol. 1803, pp. 287–296. Springer, Heidelberg (2000)
27. Merlot, L.T.G., Boland, N., Hughes, P.J., Stuckey, B.D.: A hybrid algorithm for the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 207–231. Springer, Heidelberg (2003)
28. Naji Azimi, Z.: Comparison of metaheuristic algorithms for examination timetabling problem. *Applied Mathematics and Computation* 16, 337–354 (2004)
29. Naji Azimi, Z.: Hybrid heuristics for examination timetabling problem. *Applied Mathematics and Computation* 163, 705–733 (2005)
30. Paquete, L., Stuetzle, T.: Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 413–420. Springer, Heidelberg (2003)

31. Socha, K., Knowles, J., Sampels, M.: A max–min ant system for the university course timetabling problem. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) *Ant Algorithms*. LNCS, vol. 2463, pp. 1–13. Springer, Heidelberg (2002)
32. Socha, K., Sampels, M., Manfrin, M.: Ant algorithms for the university course timetabling problem with regard to state-of-the-art. In: Raidl, G.R., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.-A., Middendorf, M. (eds.) *EvoCOP 2003*. LNCS, vol. 2611, pp. 334–345. Springer, Heidelberg (2003)
33. Stuetzle, T., Hoos, H.H.: Max–min ant systems. *Future Generation Computer Systems* 16, 889–914 (2000)
34. <http://www.cs.nott.ac.uk/~rxq/data.htm>
35. <http://www.or.ms.unimelb.edu.au/timetabling>
36. <http://www.fh-aschaffenburg.de/index.php?idlogdown>
37. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.Y.: A survey of search methodologies and automated approaches for examination timetabling. *Computer Science Technical Report No. NOTTCS-TR-2006-4*, University of Nottingham (2006)
38. Vesel, A., Zerovni, J.: How well can ants color graphs? *Journal of Computing and Information Technology (CIT)* 8, 131–136 (2000)
39. White, G.M., Xie, B.S.: Examination timetabling and tabu search with longer-term memory. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 85–103. Springer, Heidelberg (2001)
40. White, G.M., Xie, B.S., Zonjic, S.: Using tabu search with long-term memory and relaxation to create examination timetables. *European Journal of Operational Research* 153, 80–91 (2004)
41. Yang, Y., Petrovic, S.: A novel similarity measure for heuristic selection in examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 377–396. Springer, Heidelberg (2005)

# An Extensible Modelling Framework for Timetabling Problems

David Ranson<sup>1</sup> and Samad Ahmadi<sup>2</sup>

<sup>1</sup> Representational Systems Lab, Department of Informatics,  
University of Sussex, Falmer, BN1 9RH, UK  
`d.j.ranson@sussex.ac.uk`

<sup>2</sup> School of Computing, De Montfort University,  
The Gateway, Leicester, LE1 9BH, UK  
`sahmadi@dmu.ac.uk`

**Abstract.** Several modelling languages have been proposed to standardize the specification, solution and data format for timetabling problems. As of now these languages have not been adopted as standards and are seen as not simplifying the modelling process, lacking features and offering little advantage over traditional programming languages. In contrast to this approach we propose a new language-independent modelling framework for general timetabling problems based on past experience of modelling the examination timetabling problem. This framework is a work in progress but demonstrates the possibilities and convenience such a model would afford.

## 1 Introduction

Timetabling is the process of assigning events, and resources, to timeslots subject to constraints [4,21]. The feasibility of a solution for a timetabling problem is determined by the violation of *constraints* that are specified for each problem. One of the most significant application areas of timetabling is educational timetabling which is a very practical challenge faced by almost all academic institutions several times every year [8,13,18,19]. Unlike many other combinatorial optimization problems, models for educational timetabling problems change from one institution to another due to changing constraints and restrictions on resources. For this reason efforts have been made to create standard modelling languages and data formats to simplify this process.

This paper discusses the rationale for proposing a modelling framework for timetabling problems in relation to existing languages designed for timetabling. Rather than proposing a new timetabling language the idea of a standard modelling framework for timetabling problems builds on the ideas found in the existing timetabling languages and also makes use of the functionality, standardization and ease of use provided by modern object-oriented modelling frameworks. The Examination Timetabling Problem (ETP) is chosen as a significant special case of timetabling to demonstrate the practical difficulties with the existing languages and also to demonstrate the abilities of our framework.

## 2 Examination Timetabling Problem

Examination Timetabling is the problem of assigning Exams to Timeslots during a exam period respecting the given constraints [2,6,7,14]. The most important constraint for the ETP is the *clash* (or first degree student conflict) constraint which states that a student cannot be timetabled to sit more than one exam at the same time. This is an example of a *hard* constraint as a single violation renders a solution infeasible. Other examples of hard constraints are duration and room capacity constraints; i.e. exams cannot be assigned to timeslots where the duration is shorter than that of the exam.

The *consecutive* exams constraint is an example of a *soft* constraint. A consecutive constraint is violated only when a student is timetabled to sit more than one exam in immediate succession. This constraint exists in most instances of the ETP, but may not be universal. Institutions may also add their own unique constraints, such as not mixing different language exams on the same day [2]. As different institutions use very different constraints it is hard to generalize the problem in such a way that it is applicable to all cases. Any universal model for the ETP must therefore allow some flexibility in what constraints are specified. The goal in exam timetabling is to minimize the number of constraint violations over a solution. Typically a penalty is assigned to violations of soft constraints and the total cost for any solution is the sum of penalties for all the violations found.

There are many varying approaches to solving the exam timetabling problem in use at institutions and by researchers. For a comprehensive recent survey of these approaches see [14]. Some publicly available ETP data exists, such as that published by the University of Melbourne<sup>1</sup>, and has been used for benchmarking. However, these can relate to instances of the problem over a decade old, since when many universities have seen expansion in their numbers of students and courses, especially modular courses where students take exams from many different departments. There has also been some confusion over different versions of these data sets that have appeared and been used at various times [14] and issues like this need to be avoided with any future standards.

## 3 Progress Towards a Standard Format

The need for a modelling standard and standard data format has been recognized for some time and the requirements of such a standard have been discussed in detail [5]. These properties include generality, completeness, and easy translation with existing formats. It is the authors' belief that other research areas where standard formats have become the norm have benefited from increased cooperation between researchers and better benchmarking resources which have led to advances in research. Examples of this in practice include the Travelling Salesman Problem (TSPLIB) [15] and the MPL (Mathematical Programming Language [9]).

<sup>1</sup> <http://www.or.ms.unimelb.edu.au/timetabling>



A number of modelling languages have been proposed to standardize the specification, solution and data format for Timetabling Problems. In this section we briefly introduce the existing languages and provide examples of usage for one of these languages showing insight into the problems encountered.

The Standard Timetabling Language (STTL) [10,11] is a complete object-oriented functional language designed for modelling timetabling problems using set theory. STTL specifies the problem being modelled including the evaluation function, instance data and solutions.

TimeTabling Markup Language (TTML) is based on MathML which is an XML application for modelling maths formulae [12]. The goal was to create a language with the functionality of STTL but using the fashionable XML. The learning curve is steeper than that for STTL and seems overly complicated, especially for specifying the complex logic involved in these problems.

UniLang [17] is another language which has similar aims to STTL. It attempts to be a simple language easily understandable by humans as well as machines, modelling the problem by identifying subclasses of the problem and using this to guide the design. In the first aim it has largely been superseded by widely used languages such as XML. Whilst demonstrated to be capable for its purpose, UniLang does not seem as expressive as STTL or TTML.

We are unaware of any of these data formats, or any other format, being used to share timetabling data. The only known exception to this are the, previously mentioned, publicly available datasets published by Carter at the University of Melbourne. Perhaps the main reason these languages have not been adopted as standards is that they offer no advantages to the user over traditional programming languages. These somewhat idealistic languages do not simplify the modelling process, and can be restrictive in that they do not have all the features of a modern programming language, are overly complicated or appear cumbersome.

The wide variety of algorithms and software applications that use different models and data formats for timetabling problems increase the cost and difficulties of implementing new approaches. In the next section, a model for the examination timetabling problem in STTL is presented as a case study, examining some of the underlying problems with the existing approaches. A standard model for timetabling is then presented which addresses some of these issues.

## 4 Modelling the Exam Timetabling Problem in STTL

STTL [10] is a refinement of the ideas and specification previously described in [5]. As a serious attempt at creating a standard for specifying and representing timetabling problems it has the benefits of being both a standard data format and a language that can be interpreted, using Kingston's publicly available interpreter (can be downloaded from [15]), to evaluate solutions. Unlike most previous approaches STTL mixes an Object-Oriented and Functional approach to representing the resources and relationships in the problem. The result is an expressive language more than capable of modelling timetabling problems and data.

In our previous research STTL was used to model the ETP and instance data in our VAST application [16], demonstrating the functionality of the language; the model used is described in Appendix A. This ETP model was based on that Kingston [10,11] created for the High School Timetabling problem and has been made available online [15].

Our experience of STTL did make apparent some limitations of the model, due to both our design approach and issues with STTL itself. The STTL language has a learning curve and, probably because of its nature as a Functional Object Oriented language, appears quite complicated. Although set theory is one good way of specifying this kind of problem it might not be the best way from a purely modelling point of view. The sample of STTL below shows a function for finding clashes and illustrates some of the idiosyncrasies of the language:

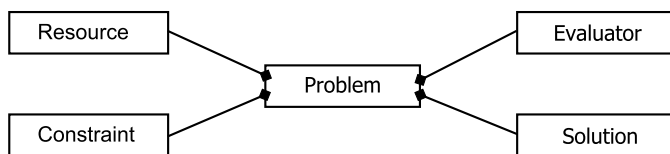
```
clashExist(e:SET[Exam]): BOOLEAN = (
  if e = {} then false
  else
    if (((head e) /= this)and ((head e).time = time) and
      ((head e).students * students))
    then true else clashExist(tail e) end
  end
)
```

In this model both timeslots and rooms are represented by corresponding classes which inherit directly from an Entity super class. However timeslots and rooms are structurally very similar, they are both containers that events are assigned to. In fact both Rooms and Timeslots can be modelled as container objects; such a model would reduce the amount of code required as both rooms and timeslots could be evaluated using the same code.

Also, in this STTL model, there are inconsistencies in the way that constraints are modelled. In some cases constraints are modelled as classes, containing all the functions for finding violations. However in other cases constraints are modelled as functions inside arbitrary classes, for example the clash constraint is implemented as a function of the Exam class (this is the reason this constraint is missing from the class diagram in Figure 2). It would be ideal if all the constraints were implemented in the same way as this would allow the existing constraints to be extended and handled in the same way by a single evaluation function.

Due to its design the STTL interpreter can be quite slow compared to other languages; the application being creating was highly interactive and needed to be very responsive. Evaluating the solution with the STTL interpreter after every change proved to be too slow for our purposes and so the evaluation function was reimplemented in Java using STTL simply as the format for data exchange.

From this experience we concluded that STTL was of most use to us as a data format for precisely specifying instances and solutions, whilst the evaluation functions and problem specifications were largely extraneous. These limitations are typical of those found with the existing approaches to modelling timetabling problems. It was found to be a relatively simple task to translate data from



**Fig. 1.** A class diagram of the classes found in the Problem model. These are the superclasses of all the other classes in the framework.

different formats into this STTL model; we created utilities to do this from the format used by Carter, and an existing Database containing the published Nottingham Data set<sup>2</sup>.

## 5 Designing a Flexible Model

The experience of using STTL and modelling timetabling problems suggested that a new, maybe simpler, approach to modelling these problems should be examined. Rather than proposing a new timetabling language we propose the idea of a standard modelling framework for timetabling problems building on the ideas found in STTL but also making use of the functionality, standardization and ease of use provided by modern Object Oriented modelling frameworks.

Our goal in this paper is to present a small and simple subset of classes, together with the relations between these classes, which are required to model the ETP, but that can be extended to model other timetabling problems. The model will be based on the structure of the problem domain rather than considering any particular approach to solving the problem or any particular implementation language. We intend to exploit the features of object-oriented programming and the UML modelling language to achieve this. Such a model would still need to conform to the requirements set out in [5] summarized as

- Generality
- Completeness of problem
- Ease of translation.

We augment these with the additional requirement, ‘ease of modelling’. This means our framework should actually make it easier to model timetabling problems, providing an incentive for adopting this flexible model over other existing formats. This is achieved in two ways:

1. Defined hierarchical framework
2. Reusable components.

It may be that this will not be the most suitable framework for every timetabling problem but our aim is to make it suitable for the vast majority of applications.

We choose an object oriented approach as this allows us use the standard inheritance mechanism to create the flexible hierarchical structure required. In

<sup>2</sup> Available from <http://www.cs.nott.ac.uk/~rxq/data>

the following examples Java terminology is used but without loss of generality. The aim of the final system is to allow code, for a visually specified timetabling model, to be generated in any programming language.

An ontology for constructing scheduling systems is proposed in [20]. The ontology proposed is structured around a constraint satisfaction model where activities are assigned resources subject to constraints. This is a good basis for modelling the timetabling problems and a similar approach is also taken in our modelling framework. Based on all these ideas we propose an extensible model built up in four layers:

1. A general abstract problem model
2. Optimization Problem model
3. General Timetabling Problem model
4. University Examination Timetabling Problem model.

Each layer builds upon the previous layer by adding problem specific resources and constraints. Once the lower layers have been implemented they can be re-used for different problems with a minimal amount of work. The functionality available at each of the lower layers is always available at the highest abstraction level, for example a constraint specified in the general timetabling problem, can also be applied to the ETP.

## 6 The Abstract General Problem Layer

The lowest level of the model that we will consider is an entirely abstract general problem defining the superclasses that the other layers in the modelling framework will extend. This layer defines classes for Resources, Constraints that need to be satisfied, Solutions and Evaluation.

Constraints are modelled as functional classes; each Constraint implements methods for evaluating a problem and its solution. The Evaluator class is responsible for aggregating the results of the constraint evaluations over the entire problem.

## 7 The Abstract General Timetabling Problem Layer

For the timetabling problem we extend our existing model, as shown in Table 1, to define the resources and constraints that are found in this domain. Event resources are assigned to Timeslot resources in order to solve the problem, Events themselves can be assigned resources. Another abstract layer actually exists between the Problem and Timetabling Problem, tentatively titled the ‘Optimization Problem’ model this provides concrete classes for the Evaluator and Constraint interfaces that allow them to return numerical values rather than abstract Objects.

The representation of time is one of the most difficult design decisions to make in a model such as this. Time is modelled as a sequence of Timeslots, with specified durations and orders, to which Events can be assigned. The CapacityTimeslot class extends this functionality to include a resource capacity that

**Table 1.** The problem, resource, solution, evaluation and constraint Classes that have been added to the Optimization and Timetabling layers of the model

Abstract Layer	Optimization Layer	Timetabling Layer
Problem	Optimization Problem	Timetabling Problem
Resource		Event Timeslot Capacity Timeslot
Solution		Timetabling Solution
Evaluator	Optimization Evaluator	Timetabling Evaluator
Constraint	Optimization Constraint	Timetabling Constraint Clash Constraint Consecutive Constraint Capacity Constraint Mixed Duration Constraint Missing Event Constraint

**Table 2.** The public interface for the TimetablingSolution class

Method	Description
assign (Event, Timeslot)	Assigns an Event to a Timeslot
unassign (Event)	Un-assigns an Event from a Timeslot
getEvents (Timeslot)	Returns list of Events assigned to Timeslot
getTimeslot(Event)	Returns Timeslot an Event is assigned to

cannot be exceeded. The timetabling solution is represented by the TimetablingSolution class which contains a list of Event, Timeslot assignments.

The TimetablingConstraint class adds functionality to find if a constraint is hard or soft; a hard constraint will render a solution infeasible if it is violated. An isFeasible() method is added to the TimetablingEvaluator which simply returns true as long as no hard constraints have been violated. If a TimetablingConstraint is violated the class needs to provide information on the specific location of the problem, for example the name and location of exams which conflict in the solution. Applications using this model could make use of this information to suggest ways to resolve such violations.

## 8 The Exam Timetabling Problem Layer

To complete our Exam Timetabling model the Classes required to implement the ETP layer are added to the model by extending the existing classes. The new resources added to the framework here are shown in Table 3. Note that the Solution Class does not need to be changed to implement this layer. The only classes needed are those that map the abstract Timetabling problem to the

**Table 3.** The Classes required to implement the Exam Timetabling layer of the model. Classes on the same row extend the same superclass, shown in the first column.

Abstract Layer	Timetabling Layer	Exam Timetabling Layer
Problem	Timetabling Problem	ETP Problem
Resource	Event CapacityTimeslot	Exam RoomTimeslot Student
Solution	Timetabling Solution	
Evaluator	Timetable Evaluator	ETP Evaluator

concrete Exam Timetabling problem. It is envisioned that further timetabling problems could be modelled using this framework with similar ease. Our work modelling other problems, including the High School Timetabling problem, has already begun as well as the implementation of Solver Classes which can be used with the framework. The complete API documentation describing all the Classes in the complete ETP model has been published online [15] and we expect to make our Java implementation available in the near future.

## 9 Exchange and Persistence of Timetabling Data

Whilst the primary aim of this paper was not to supply a new data format for timetabling problems, our framework does provide us with such a format. The structure presented in our model can be maintained in an XML document using tags formed from the classes in our model.

Using this XML representation as a format for timetabling data exchange and persistence has certain advantages. Firstly, for most modern object-oriented languages, it can be created very easily using XML serialization and deserialization of the Problem object. Unlike some other formats the resulting documents are both human readable and completely self-documenting. Although any problem modelled within our framework can be represented in this way a lot of the complexity that exists in other formats is reduced because only the data is stored, not any of the logic. This does mean that the problem is only completely specified when combined with the modelling framework; however as long as there is agreement on how the Constraints are implemented this is not a serious issue.

The published Carter timetabling data, previously referred to, has been converted into this new format and an example application that demonstrates the use of our modelling framework to load and evaluate this data is available online [15]. In most cases it is a relatively trivial task to convert existing data into this format and it is our intention to set up a repository where data in this format can be shared. This repository, in conjunction with other similar efforts [3], could be used to bring ease in using real-world timetabling data for use by different researchers as well as maintaining the integrity and consistency of the data.

## 10 Future Work

In this paper an attempt to design an extensible modelling framework, with the aim of simplifying the modelling process for timetabling problems, is reported and the applicability of this approach is demonstrated to model the ETP. However, to demonstrate the extensibility, it will be necessary to show that the model works for other timetabling applications, as has started with High School Timetabling, and that the same design consistency can be applied across different problems in this domain.

One future goal is the implementation of a graphical tool which can be used to assemble the reusable components of our framework, making use of an online repository, in order to further simplify the modelling process of new problems. Such an application would allow non programmers to easily model timetabling problems, making use of the reusable Resources and Constraints or adding new ones to the central repository as required.

## 11 Concluding Remarks

The aim of this paper has partly been to reignite discussion on the issue of ‘Standard Timetabling Languages’ but mainly to promote our ideas on a different approach to this topic and how these problems could be modelled in line with modern programming paradigms.

Unlike other approaches we have deliberately shied away from advocating a particular programming language (apart from for the purposes of demonstrating our own implementation) as we believe this is best decided by the capabilities of the user. All mainstream languages are capable of modelling problems in this domain. Trying to form consensus around a standardized language is always difficult but focusing on this when such a language is not required can cause discussion to stagnate and limit progress.

## References

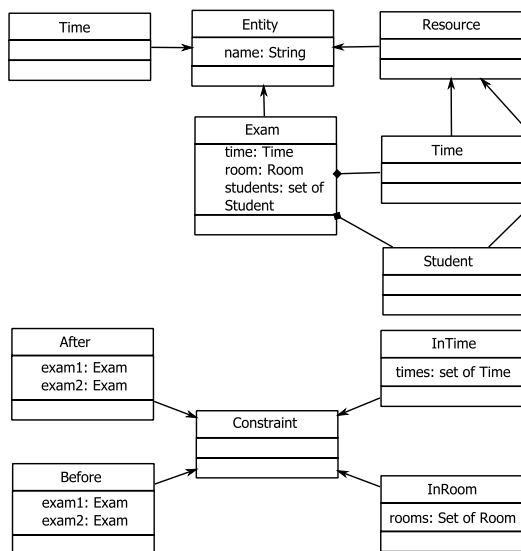
1. Bixby, B., Reinelt, G.: TSPLIB a library of travelling salesman and related problem instances (1995), <http://softlib.rice.edu/tsplib.html>
2. Burke, E.K., Elliman, D.G., Ford, P.H., Weare, R.F.: Examination timetabling in British universities – a survey. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 76–90. Springer, Heidelberg (1996)
3. Burke, E. K., McCollum, B.: Examination timetabling: a new formulation (Abstract). In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, August (2006) 373–375
4. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002)
5. Burke, E.K., Kingston, J.H., Pepper, P.A.: A standard data format for timetabling instances. In: Burke, E.K., Carter, M. (eds.) *PATAT 1997*. LNCS, vol. 1408, pp. 213–222. Springer, Heidelberg (1998)

6. Burke, E.K., de Werra, D., Kingston, J.: Applications to timetabling. In: Gross, J., Yellen, J. (eds.) *The Handbook of Graph Theory*, pp. 445–474. Chapman Hall/CRC Press, London (1997)
7. Carter, M.W.: A survey of practical applications of examination timetabling algorithms. *Operations Research* 34, 193–202 (1986)
8. Carter, M.W.: Timetabling. In: Gass, S., Harris, C.M. (eds.) *Encyclopedia of Operations Research and Management Science*, pp. 833–836. Kluwer, Dordrecht (2001)
9. Dantzig, G.B., Eisenstat, S.C., Magnanti, T.L., Maier, S.F., McGrath, M.B.: The mathematical programming language (MPL). In: *Proceedings of the 1971 26th Annual Conference*, pp. 278–283. ACM Press, New York (1971)
10. Kingston, J.H.: Modelling timetabling problems with STTL. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 309–321. Springer, Heidelberg (2001)
11. Kingston, J.H.: A user's guide to the STTL timetabling language version 1.0. <http://www.it.usyd.edu.au/~jeff/ttsttl1.ps>
12. Ozcan, E.: Towards an XML based standard for timetabling problems: TTML. In: *MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, Nottingham, pp. 566–569 (August 2003)
13. Petrovic, S., Burke, E.K.: University timetabling. In: Leung, J. (ed.) *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL (2004)
14. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.: A survey of search methodologies and automated approaches for examination timetabling. University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-2006-4 (2006)
15. Ranson, D.: Extensible modelling framework for timetabling optimisation problem website. <http://www.informatics.sussex.ac.uk/users/djr23/emfop/>
16. Ranson, D., Cheng, P.C.H.: Graphical tools for heuristic visualization. In: *MISTA. The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications*, New York, pp. 658–668 (July 2005)
17. Reis, L.P., Oliveira, E.: A language for specifying complete timetabling problems. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 322–341. Springer, Heidelberg (2001)
18. Rudova, H., Murray, K.: University course timetabling with soft constraints. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 310–328. Springer, Heidelberg (2003)
19. Schaerf, A.: A survey of automated timetabling. *Artificial Intelligence Review* 13, 87–127 (1999)
20. Smith, S., Becker, M.: An ontology for constructing scheduling systems. In: *Working Notes from 1997 AAAI Spring Symposium on Ontological Engineering*, Stanford, CA, AAAI Press, Menlo Park, CA (March 1997)
21. Wren, A.: Scheduling, timetabling and rostering – a special relationship? In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 46–75. Springer, Heidelberg (1996)

## A The Exam Timetabling Model in STTL

Each STTL model is made up of three components, normally split into separate files. The problem file contains the STTL code for modelling the problem, constraints, and the evaluation function. An instance file contains concrete data





**Fig. 2.** Class Diagram of STTL ETP model. Arrows represent an ‘is a’ inheritance relationship between two classes whilst diamonds represent a ‘1-many’ relationship. Not all the Constraint classes are illustrated.

for an instance of the problem and solution files contains values for the solution variables found in the problem file.

All the Classes needed to model the ETP in an STTL Problem file are illustrated in Figure 2. The Entity class is a superclass for all the resources found in the problem, whilst the Constraint classes are used by the evaluation function. The Before and After constraints are order precedence relations whilst InRoom and InTime constrain which times and rooms an exam can be assigned.

# An Experimental Study on Hyper-heuristics and Exam Timetabling

Burak Bilgin, Ender Özcan, and Emin Erkan Korkmaz

Artificial Intelligence Laboratory (ARTI), Yeditepe University,  
Department of Computer Engineering, 34755 Kadıköy/Istanbul, Turkey  
{bbilgin, eozcan, ekorkmaz}@cse.yeditepe.edu.tr

**Abstract.** Hyper-heuristics are proposed as a higher level of abstraction as compared to the metaheuristics. Hyper-heuristic methods deploy a set of simple heuristics and use only non-problem-specific data, such as fitness change or heuristic execution time. A typical iteration of a hyper-heuristic algorithm consists of two phases: the heuristic selection method and move acceptance. In this paper, heuristic selection mechanisms and move acceptance criteria in hyper-heuristics are analyzed in depth. Seven heuristic selection methods and five acceptance criteria are implemented. The performance of each selection and acceptance mechanism pair is evaluated on 14 well-known benchmark functions and 21 exam timetabling problem instances.

## 1 Introduction

The term hyper-heuristic refers to a recent approach used as a search methodology [2, 3, 5, 11, 20]. It represents a higher level of abstraction than most of the uses of metaheuristic methods in the literature. Hyper-heuristics involve an iterative strategy that chooses a heuristic to apply to a candidate solution of the problem at hand, at each step. The properties of hyper-heuristics are discussed in [3]. An iteration of a hyper-heuristic can be subdivided into two parts: heuristic selection and move acceptance. In the hyper-heuristic literature, several heuristic selection and acceptance mechanisms are used [2, 3, 5, 11, 20]. However, no comprehensive study exists that compares the performance of these different mechanisms in depth.

Timetabling problems are real-world constraint optimization problems. Due to their NP-complete nature [16], the traditional approaches might fail to generate a solution to a timetabling problem instance. The timetabling problems require assignment of *time-slots* (periods) and possibly some other resources to a set of events, subject to a set of constraints. Numerous researchers deal with different types of timetabling problems based on different types of constraints utilizing a variety of approaches. *Employee timetabling*, *course timetabling* and *examination timetabling* are the research fields that attract the most attention. In this paper, seven heuristic selection methods and five different acceptance criteria are analyzed in depth. Their performance is measured on well-known

benchmark functions. Moreover, 35 hyper-heuristics, generated by coupling all heuristic selection methods and all acceptance criteria with each other, are evaluated on a set of 21 exam timetabling benchmark problem instances, including the Carter's [10] and Ozcan's [25] benchmarks.

The remainder of this paper is organized as follows. In Section 2 background is provided including the hyper-heuristics, benchmark functions and exam timetabling. Experimental settings and results for benchmarks are given in Section 3. The hyper-heuristic experiments on exam timetabling are presented in Section 4. Finally, the conclusions are discussed in Section 5.

## 2 Preliminaries

### 2.1 Hyper-heuristics

Hyper-heuristic methods are described in [3] as an alternative method to techniques that work directly on solution spaces. Most applications of Metaheuristics are 'problem-specific' solution methods, which require knowledge and experience about the problem domain and properties. Metaheuristics are mostly developed for a particular problem and require fine tuning of parameters. Therefore, they can be developed and deployed only by experts who have the sufficient knowledge and experience on the problem domain and the meta-heuristic search method. Hyper-heuristics, on the other hand, are developed to be more general optimization methods, which can be applied to more optimization problems easily. Hyper-heuristics can be considered as black box systems, which take the problem instance and several low-level heuristics as input and which can produce the result independent of the problem characteristics. In this concept, hyper-heuristics use only non-problem-specific data provided by each low-level heuristic in order to select and apply them to the candidate solution [3, 5, 11].

The selection mechanisms in the hyper-heuristic methods were emphasized in the initial phases of the research period. Cowling et al. [11] proposed three types of low-level heuristic selection mechanisms to be used in hyper-heuristics; which are *Simple*, *Greedy* and *Choice Function*. There are four types of *Simple* heuristic selection mechanisms. *Simple Random* mechanism chooses a low-level heuristic at a time randomly. *Random Descent* mechanism chooses a low-level heuristic randomly and applies it repeatedly as long as it produces improving results. *Random Permutation* mechanism creates an initial permutation of the low-level heuristics and at each iteration applies the next low-level heuristic in the permutation. *Random Permutation Descent* mechanism is the same as *Random Permutation* mechanism, except that it applies the low-level heuristic in turn repeatedly as long as it produces improving results. *Greedy* method calls each low-level heuristic at each iteration and chooses the one that produces the most improving solution. *Choice Function* is the most complex one. It analyzes both the performance of each low-level heuristic and each pair of low-level heuristics. This analysis is based on the improvement and execution time. This mechanism also considers the overall performance. It attempts to focus the search as long as the improvement rate is high and broadens the search if the improvement

rate is low. For each of these low-level heuristic selection mechanisms two simple acceptance criteria are defined. These are *AM*, where all moves are accepted and *OI* where only improving moves are accepted [11].

Burke et al. [5] proposed a *Tabu-Search* heuristic selection method. This mechanism ranks low-level heuristics. At the beginning of the run each heuristic starts the execution with the minimum ranking. Every time a heuristic produces an improving movement its rank is increased by a positive reinforcement rate. The rank of the heuristics cannot exceed a predetermined maximum value. Whenever a heuristic cannot make an improving move; its rank is decreased by a negative reinforcement learning rate. Similarly the rank of a heuristic cannot be decreased to a value less than a predetermined minimum value. In the case of worsening moves, the heuristic is also added to the tabu list. Another parameter is the tabu duration which sets the maximum number of iterations a low-level heuristic can stay in the tabu list.

Burke et al. [8] introduce a simple generic hyper-heuristic which utilizes constructive heuristics (graph coloring heuristics) to tackle timetabling problems. A tabu-search algorithm chooses among permutations of constructive heuristics according to their ability to construct complete, feasible and low-cost timetables. At each iteration of the algorithm, if the selected permutation produces a feasible timetable, a deepest descent algorithm is applied to the obtained timetable. Burke et al. used this hyper-heuristic method in exam and university course timetabling problem instances. The proposed method worked well on the related benchmark problem instances [8].

Burke et al. [9] proposed a case-based heuristic selection approach. A knowledge discovery method is employed to find the problem instances and situations where a specific heuristic has a good performance. The proposed method also explores the similarities between the problem instance and the source cases, in order to predict the heuristic that will perform best. Burke et al. applied the Case-Based Heuristic Selection Approach to exam and university course timetabling [9].

Ayob and Kendall [2] emphasized the role of the acceptance criterion in the hyper-heuristic. They introduced the *Monte Carlo Hyper-heuristic* which has a more complex acceptance criterion than *AM* or *OI* criteria. In this criterion, all of the improving moves are accepted and the non-improving moves can be accepted based on a probabilistic framework. Ayob and Kendall defined three probabilistic approaches to accept the non-improving moves. The first approach, named *Linear Monte Carlo* (LMC), uses a negative linear ratio of the probability of acceptance to the fitness worsening. The second approach, named *Exponential Monte Carlo* (EMC), uses a negative exponential ratio of the probability of acceptance to the fitness worsening. The third approach, named *Exponential Monte Carlo with Counter* (EMCQ), is an improvement over EMC. Again, the probability of accepting worsening moves decreases as the time passes. However if no improvement can be achieved over a series of consecutive iterations then this probability starts increasing again. As the heuristic selection mechanism, they all use a simple random mechanism [2].

Kendall and Mohamad [20] introduced another hyper-heuristic method which also focuses on acceptance criterion rather than selection method. They used the *Great Deluge Algorithm* as the acceptance criterion and *Simple Random* as the heuristic selection method. In the *Great Deluge Algorithm* initial fitness is set as the initial level. At each step, the moves which produce fitness values less than the level are accepted. At each step the level is also decreased by a factor [20].

Rattadilok et al. [29] presented a research on the choice function hyper-heuristics, generalized low-level heuristics, and utilization of parallel computing environments for hyper-heuristics. An abstract low-level heuristic model is proposed which can be easily implemented to be a functional low-level heuristic tackling a specific problem type. The choice function hyper-heuristic and the low-level heuristics are improved to evaluate a broader range of the data. Two types of distributed hyper-heuristic approaches are introduced. The first approach is a single hyper-heuristic, multiple low-level heuristics which are executed on different nodes and focus on different areas of the timetable. The second approach utilizes multiple hyper-heuristics each of which work on a different node. In this approach, hyper-heuristics collaborate during the execution [29].

According to this survey it is concluded that several heuristic selection methods and acceptance criteria are introduced for the hyper-heuristic framework. Each pair of heuristic selection and acceptance mechanism can be used as a different hyper-heuristic method. Despite this fact, such combinations have not been studied in the literature. In this study, seven heuristic selection mechanisms, which are Simple Random, Random Descent, Random Permutation, Random Permutation Descent, Choice-Function, Tabu-Search and Greedy heuristic selection mechanisms, are implemented. For each heuristic selection method five acceptance criteria: *AM*, *OI*, *IE*, a *Great Deluge* and a *Monte Carlo* are used. As a result a broad range of hyper-heuristic variants are obtained. These variants are tested on mathematical objective functions and exam timetabling problems.

## 2.2 Benchmark Functions

Well-defined problem sets are useful to measure the performance of optimization methods such as genetic algorithms, memetic algorithms and hyper-heuristics. Benchmark functions which are based on mathematical functions or bit strings can be used as objective functions to carry out such tests. The characteristics of these benchmark functions are explicit. The difficulty levels of most benchmark functions are adjustable by setting their parameters. In this study, 14 different benchmark functions are chosen to evaluate the hyper-heuristics.

The benchmark functions presented in Table 1 are continuous functions, and *Royal Road Function*, *Goldberg's 3 bit Deceptive Function* [17, 18] and *Whitley's 4 bit Deceptive Function* [31] are discrete functions. Their deceptive nature is due to the large Hamming distance between the global optimum and the local optima. To increase the difficulty of the problem  $n$  dimensions of these functions can be combined by a summation operator.

The candidate solutions to all the continuous functions are encoded as bit strings using Gray code. The properties of the benchmark functions are

**Table 1.** Properties of benchmark functions, *lb* indicates the lower bound, *ub* indicates the upper bound of the search space, *opt* indicates the global optimum in the search space

Function [Source]	<i>lb</i>	<i>ub</i>	<i>opt</i>	Continuity	Modality
Sphere [13]	-5.12	5.12	0	Continuous	Unimodal
Rosenbrock [13]	-2.048	2.048	0	Continuous	Unimodal
Step [13]	-5.12	5.12	0	Continuous	Unimodal
Quartic [13]	-1.28	1.28	1	Continuous	Multimodal
Foxhole [13]	-65.536	65.536	1	Continuous	Multimodal
Rastrigin [28]	-5.12	5.12	0	Continuous	Multimodal
Schwefel [30]	-500	500	0	Continuous	Multimodal
Griewangk [19]	-600	600	0	Continuous	Multimodal
Ackley [1]	-32.768	32.768	0	Continuous	Multimodal
Easom, [15]	-100	100	-1	Continuous	Unimodal
Rotated Hyperellipsoid[13]	-65.536	65.536	0	Continuous	Unimodal
Royal Road [23]	-	-	0	Discrete	-
Goldberg [17, 18]	-	-	0	Discrete	-
Whitley [31]	-	-	0	Discrete	-

presented in Table 1. The modality property indicates the number of optima in the search space (i.e. between bounds). Unimodal benchmark functions have a single optimum. Multimodal benchmark functions contain more than one optimum in their search space. Such functions contain at least one additional local optimum in which a search method can get stuck.

### 2.3 Exam Timetabling

Burke et al. [4] applied a light or a heavy mutation, randomly selecting one, followed by a hill climbing method in a memetic algorithm framework for solving exam timetabling problems. Burke et al. [6] provided a survey of timetabling problems at UK universities. Investigation of various combinations of Constraint Satisfaction Strategies with GAs for solving exam timetabling problems can be found in [21]. Paquete et al. [26] employed a multi-objective evolutionary algorithm (MOEA) based on Pareto ranking for solving the exam timetabling problem in the Unit of Exact and Human Sciences at the University of Algarve. Two objectives were determined: to minimize the number of conflicts within the same *group* and the conflicts between different *groups*. Wong et al. [32] used a GA utilizing a non-elitist replacement strategy to solve a single exam timetabling problem at an École de Technologie Supérieure. After genetic operators were applied, violations were fixed in a hill climbing procedure.

Carter et al. [10] applied different heuristic orderings based on graph coloring. Their experimental data became one of the commonly used exam timetabling benchmarks. Gaspero and Schaerf [14] analyzed the tabu search approach using graph coloring based heuristics. Merlot et al. [22] explored a hybrid approach for

solving the exam timetabling problem that produces an initial feasible timetable via constraint programming. The method then applies simulated annealing with hill climbing to improve the solution. Petrovic et al. [27] introduced a case-based reasoning system to create initial solutions to be used by the Great Deluge algorithm. Burke et al. [7] proposed a general and fast adaptive method that arranges the heuristic to be used for ordering exams to be scheduled next. Their algorithm produced comparable results on a set of benchmark problems with the current state of the art. This study is useful as a guide for formulating the constraints and objective function to be used in a timetabling problem. Ozcan and Ersoy [25] used a violation-directed adaptive hill climber within a memetic algorithm to solve the exam timetabling problem. A Java tool named FES is introduced by Ozcan in [24] which utilizes XML as input/output format.

The exam timetabling problem can be formulated as a constraint optimization problem by a 3-tuple  $(V, D, C)$ .  $V$  is a finite set of examinations,  $D$  is a finite set of domains of variables, and  $C$  is a finite set of constraints to be satisfied. In this representation a variable stands for an exam schedule of a course. Exam timetabling involves a search for a solution, where values from domains (timeslots) are assigned to all variables while satisfying all the constraints.

The set of constraints for the exam timetabling problem differs from institution to institution. In this study, three constraints are defined and used as described in [25]:

1. A student cannot be scheduled to two exams at the same time slot.
2. If a student is scheduled to two exams in the same day, these should not be assigned to consecutive timeslots.
3. The total capacity for a timeslot cannot be exceeded.

### 3 Hyper-heuristics for Benchmark Functions

#### 3.1 Benchmark Function Heuristics

Six heuristics were implemented to be used with hyper-heuristics on benchmark functions. Half of these are hill-climbing methods and the remaining half are mutational operators combined with a hill climber.

*Next Ascent Hill Climber* produces number of bits times iterations at each heuristic call. Starting from the most significant bit, at each iteration it inverts the next bit in the bit string. If there is a fitness improvement, the modified candidate solution is accepted as the current candidate solution [23]. *Davis' Bit Hill Climber* is the same as Next Ascent Hill Climber but it does not modify the bit sequentially but in the sequence of a randomly determined permutation [12]. *Random Mutation Hill Climber* chooses a bit randomly and inverts it. Again, the modified candidate solution becomes the current candidate solution, if the fitness is improved. This step is repeated for total number of bits in the candidate solution times iterations at each heuristic call [23].

Mutational heuristics are *Swap Dimension*, *Dimensional Mutation* and *Hypermutation*. *Swap Dimension* heuristic randomly chooses two different dimensions in the candidate solution and swaps them. *Dimensional Mutation* heuristic

**Table 2.** Average ranking of each selection method on each problem: CF, Choice Function; SR, Simple Random; RD, Random Descent; RP, Random Permutation; RPD, Random Permutation Descent; Tabu, Tabu Search; GR, Greedy

Name	CF	SR	RD	RP	RPD	TABU	GR
Sphere	7.0	7.0	24.5	14.0	24.5	24.5	24.5
Rosenbrock	20.2	22.0	16.0	23.8	16.0	16.0	12.0
Step	17.7	17.7	17.7	18.9	17.7	17.7	18.6
Quartic <i>w/ noise</i>	17.9	17.9	17.9	17.9	17.9	17.9	18.6
Foxhole	15.7	15.7	15.7	19.3	15.7	15.7	28.2
Rastrigin	17.9	17.5	18.5	17.3	18.5	17.7	18.6
Schwefel	17.0	17.0	18.8	17.0	18.8	18.8	18.6
Griewangk	11.8	17.2	17.2	17.2	17.2	17.2	28.2
Ackley	16.5	16.5	16.5	23.5	16.5	16.5	20.0
Easom	16.0	16.0	21.7	16.0	21.7	21.7	12.9
Rotated Hyperellipsoid	20.4	21.2	13.4	21.6	14.8	19.8	15.6
Royal Road	16.8	17.6	17.1	17.4	17.1	17.8	22.2
Goldberg	18.6	19.3	16.6	19.4	17.4	16.1	18.6
Whitley	17.9	17.9	17.9	17.9	17.9	17.9	18.6

randomly chooses a dimension and inverts each bit in this dimension with the probability 0.5. *Hypermutation* randomly inverts each bit in the candidate solution with the probability 0.5. To improve the quality of candidate solutions obtained from these mutational heuristics, Davis’s Bit Hill Climbing is applied.

### 3.2 Experimental Settings

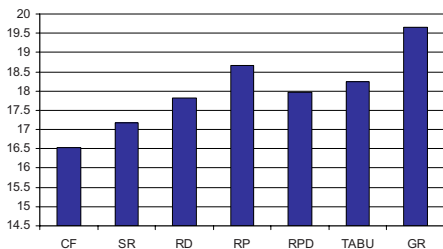
The experiments are performed on Pentium IV, 2 GHz Linux machines with 256 Mb memory. Fifty runs are performed for each hyper-heuristic and problem instance pair. For each problem instance, a set of fifty random initial configurations are created. Each run in an experiment is performed starting from the same initial configuration. The experiments are allowed to run for 600 CPU seconds. If the global optimum of the objective function is found before the time limit is exhausted, then the experiment is terminated.

The candidate solutions are encoded as bit strings. The continuous functions in benchmark set are encoded in Gray code. The discrete functions have their own direct encoding. The Foxhole function has default dimension of 2. The default number of bits per dimension parameter is set to 8, 3, and 4 for the Royal Road, Goldberg, and Whitley functions respectively. The rest of the functions have 10 dimensions and 30 bits are used to encode the range of a variable.

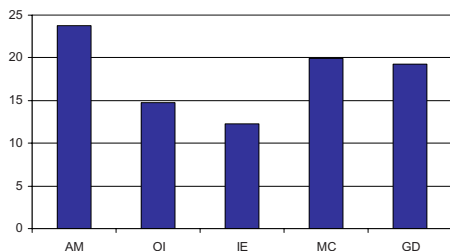
### 3.3 Experimental Results

The experimental results of performance comparison of 35 heuristic selection–acceptance criteria combinations on 14 different benchmark functions are statistically evaluated. For each benchmark function the combinations are sorted





**Fig. 1.** Average ranking of each selection method on all problem instances



**Fig. 2.** Average ranking of each acceptance criterion on all problem instances

according to their performance. The average number of fitness evaluations needed to converge to global optimum is used as the performance criterion for the experiments with 100% success rate. The average best fitness reached is used for the experiments with success rates lower than 100%. The performances are evaluated statistically using a  $t$ -test. Each combination has been given a ranking. Confidence interval is set to 95% in the  $t$ -test to determine significant performance variance. The combinations that do not have significant performance variances are grouped together and have been given the same ranking. The average rankings of heuristic selection methods and move acceptance criteria are calculated to reflect their performance. In Table 2, average rankings for the heuristic selection methods are provided on each problem. The averages are obtained by testing the selection methods on each acceptance criterion. In Table 3, average rankings of acceptance criteria are given where the averages are obtained by testing acceptance criteria on each selection method this time. Lower numbers in these tables denote a higher placement in the ranking and indicate better performance. The average ranking of each selection method on all of the functions is shown in Figure 1, and the average ranking of each acceptance criterion on all of the functions is shown in Figure 2.

No heuristic selection and acceptance criterion couple came out to be a winner on all of the benchmark functions. *Choice Function* performs well on *Sphere* and *Griewangk* functions. *Simple Random* performs well on *Sphere Function*. *Random Descent* and *Random Permutation Descent* perform well on *Rotated*

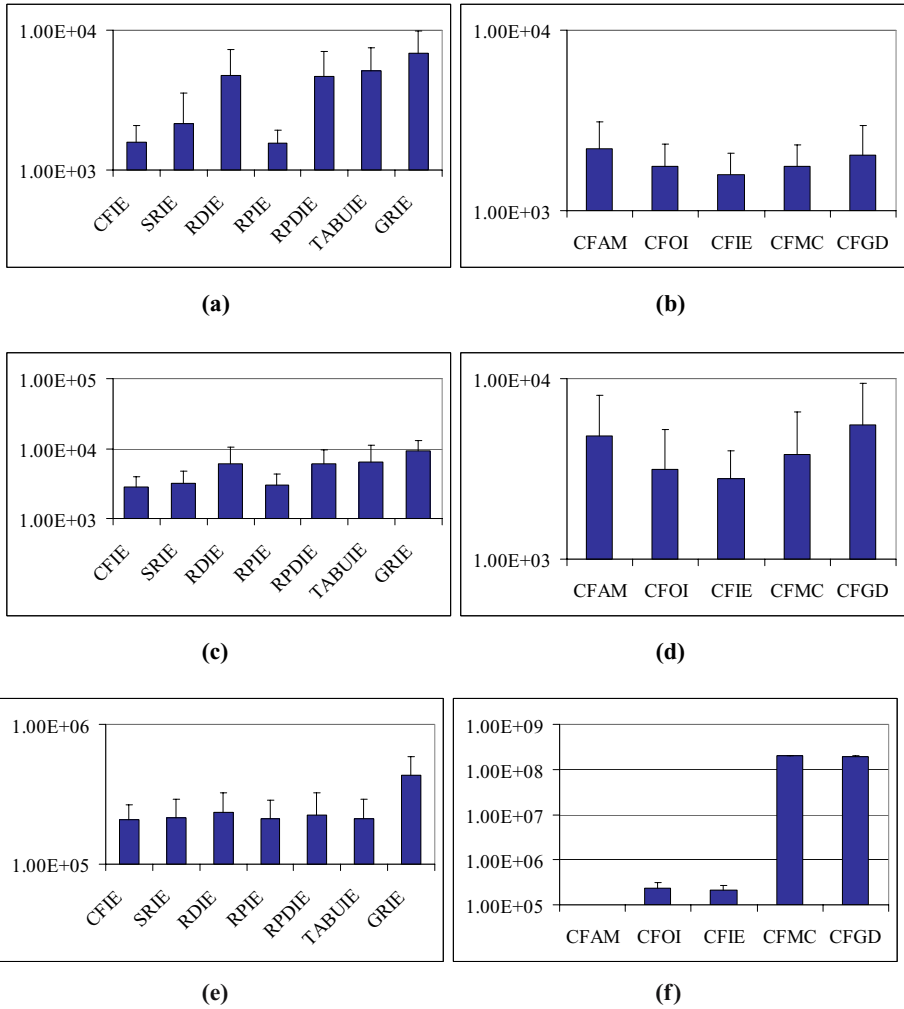
**Table 3.** Average ranking of each acceptance criterion on each problem: AM, All Moves Accepted; OI, Only Improving Moves Accepted; IE, Improving and Equal Moves Accepted; MC, Monte Carlo Acceptance Criterion; GD, Great Deluge Acceptance Criterion

Name	AM	OI	IE	MC	GD
Sphere	19.5	17.0	17.0	17.0	19.5
Rosenbrock	23.8	12.0	16.0	23.8	16.0
Step	29.1	18.6	17.7	18.9	17.7
Quartic <i>w/noise</i>	29.1	17.4	14.5	14.5	14.5
Foxhole	12.4	27.7	26.5	11.1	12.4
Rastrigin	29.1	10.6	7.6	23.9	18.8
Schwefel	29.1	10.6	7.6	22.6	20.1
Griewangk	11.9	27.7	26.5	11.9	11.9
Ackley	19.0	19.0	16.5	16.5	19.0
Easom	23.3	11.6	8.5	23.3	23.3
Rotated Hyperellipsoid	25.1	11.7	8.8	22.4	22.6
Royal Road	28.1	10.6	7.6	23.0	20.7
Goldberg	29.1	10.6	7.6	22.4	20.4
Whitley	23.9	10.6	7.6	23.9	23.9

*Hyperellipsoid Function.* Greedy performs well on *Rosenbrock Function*. The performance variances of heuristic selection methods on remaining functions were not as significant as these cases. Choice Function performs slightly better than remaining selection methods on average. *IE* acceptance criterion performs well on *Rastrigin*, *Schwefel*, *Easom*, *Rotated Hyperellipsoid*, and discrete deceptive functions. *OI* acceptance criterion performs well on *Rosenbrock Function*. The *MC* acceptance criterion performs well on *Foxhole Function*. The *IE* acceptance criterion indicates a significantly better performance than the remaining acceptance criteria on average.

In Figure 3 average number of evaluations to converge to global optimum by a selected subset of hyper-heuristics is depicted on a subset of benchmark functions, which are *Sphere*, *Ackley* and *Goldberg Functions*. Figures 3(a), (c), and (e) show the performance comparison of the heuristic selection methods using *IE* acceptance criterion for *Sphere*, *Ackley* and *Goldberg Functions* respectively and Figures 3(b), (d), and (f) the performance comparison of the acceptance criteria using the Choice Function heuristic selection method for *Sphere*, *Ackley* and *Goldberg Functions* respectively. Lower average number of evaluations means faster convergence to the global optimum and indicates better performance.

For *Sphere Model*, distinct performance variances are observed between heuristic selection methods in Figure 3(a). On the other hand, the difference is not so prominent between acceptance criteria in Figure 3(b). Figure 3(a) shows that the Random Permutation and Choice Function heuristic selection methods achieved faster convergence than the remaining selection methods. In Figures 3(c) and (d) it can be observed that the Choice Function heuristic selection method and *IE* acceptance criterion accomplished a faster convergence to global



**Fig. 3.** Average number of evaluations to converge to global optimum of hyper-heuristics consisting of all heuristic selection methods using IE acceptance criterion on (a) *Sphere Model* function, (c) *Ackley Function*, (e) *Goldberg Function*, and average number of evaluations to converge to global optimum of hyper-heuristics consisting of Choice Function heuristic selection method and all acceptance criteria on (b) *Sphere Model* function, (d) *Ackley Function*, (f) *Goldberg Function*.

optimum on *Ackley Function*. Figures 3(e) and (f) show that the Choice Function heuristic selection method and IE acceptance criterion performed best on *Goldberg's Function*. Figure 3(f) shows that the performance variances between different acceptance criteria are enormous on the same function. Also, the AM

**Table 4.** Parameters and properties of the exam timetabling problem instances

Instance	Exams	Students	Enrollment	Density	Days	Capacity
Carf92	543	18419	54062	0.14	12	2000
Cars91	682	16925	59022	0.13	17	1550
Earf83	181	941	6029	0.27	8	350
Hecs92	81	2823	10634	0.20	6	650
Kfus93	486	5349	25118	0.06	7	1955
Lsef91	381	2726	10919	0.06	6	635
Purs93	2419	30032	120690	0.03	10	5000
Ryes93	486	11483	45051	0.07	8	2055
Staf83	139	611	5539	0.14	4	3024
Tres92	261	4360	14901	0.18	10	655
Utas92	622	21267	58981	0.13	12	2800
Utes92	184	2749	11796	0.08	3	1240
Yorf83	190	1125	8108	0.29	7	300
Yue20011	140	559	3488	0.14	6	450
Yue20012	158	591	3706	0.14	6	450
Yue20013	30	234	447	0.19	2	150
Yue20021	168	826	5757	0.16	7	550
Yue20022	187	896	5860	0.16	7	550
Yue20023	40	420	790	0.19	2	150
Yue20031	177	1125	6716	0.15	6	550
Yue20032	210	1185	6837	0.14	6	550

acceptance criterion cannot reach the global optimum on *Goldberg's Function* and no average number of evaluations to converge to global optimum value is shown for this criterion in the same figure.

## 4 Hyper-heuristics for Solving Exam Timetabling Problems

### 4.1 Exam Timetabling Problem Instances and Settings

Carter's Benchmark [10] and Yeditepe University Faculty of Architecture and Engineering [25] data sets are used for the performance comparison of the hyper-heuristics. The characteristics of the experimental data are illustrated in Table 4.

Hyper-heuristics consisting of *Simple Random*, *Random Descent*, *Tabu Search*, *Choice Function*, and *Greedy* heuristic selection mechanisms and all the acceptance criteria, described in Section 2.1, are tested with each benchmark exam timetabling problem instance. The fitness function used for solving the exam timetabling problem takes a weighted average of the number of constraint violations. The fitness function is multiplied by  $-1$  to make the problem a minimizing problem:

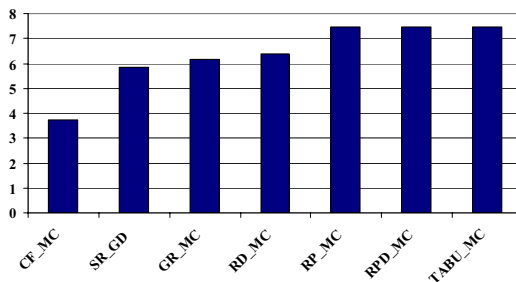
$$F(T) = \frac{-1}{1 + \sum_{\forall i} w_i g_i(T)}. \tag{1}$$

In Equation (1),  $w_i$  indicates the weight associated to the  $i$ th constraint,  $g_i$  indicates the number of violations of  $i$ th constraint for a given schedule  $T$ . The value 0.4 is used as the weight for the first and the third constraint and 0.2 for the second constraint, as explained in Section 2.3.

### 4.2 Heuristics for Exam Timetabling

Candidate solutions are encoded as an array of timeslots where each locus represents an exam to be scheduled. Four heuristics are implemented to be used with the hyper-heuristics for solving an exam timetabling problem. Three of these heuristics utilize a tournament strategy for choosing a timeslot to reschedule a given exam to improve a candidate solution based on a constraint type, while the last one is a mutation operator. Heuristics for the constraints (i) and (ii) work similarly. Each improving heuristic targets a different conflict. Both heuristics randomly choose a predetermined number of exams and select the exam with the highest number of targeted conflicts among these. Also a predetermined number of timeslots are randomly chosen and the number of targeted conflicts are checked when the exam is assigned to that timeslot. The timeslot with the minimum number of targeted conflicts is then assigned to the selected exam.

The heuristic which targets the capacity conflicts (iii) randomly chooses a predetermined number of timeslots and selects the timeslot with the maximum capacity conflict among these. A predetermined number of exams that are scheduled to this timeslot are chosen randomly and the exam that has the most attendants is selected among them. Again a group of timeslots are chosen randomly and the timeslot with the minimum number of attendants is assigned to the selected exam. The mutational heuristic passes over each exam in the array and assigns a random timeslot to the exam with a predetermined probability ( $1/\text{number of courses}$ ).



**Fig. 4.** Top seven heuristic selection method and acceptance criterion combinations considering the average ranking over all exam timetabling problem instances

**Table 5.** Average best fitness values for best performing heuristic selection–acceptance criterion combinations on each problem instance: AM, All Moves Accepted; OI, Only Improving Moves Accepted; IE, Improving and Equal Moves Accepted; MC, Monte Carlo acceptance criterion; GD, Great Deluge acceptance criterion

Instance	(Av. B. Fit., SD)	H. Heuristic Alg.
Carf92	(-1.02E-02, 1.18E-03)	TABU_IE *
Cars91	(-1.93E-01, 1.20E-01)	TABU_IE *
Earf83	(-7.27E-03, 4.94E-04)	CF_MC
Hecs92	(-2.19E-02, 2.43E-03)	CF_MC *
Kfus93	(-3.40E-02, 4.30E-03)	SR_GD
Lsef91	(-1.42E-02, 1.38E-03)	CF_MC
Purs93	(-1.41E-03, 6.98E-05)	SR_IE
Ryes93	(-1.08E-02, 1.37E-03)	CF_MC
Staf83	(-2.68E-03, 1.04E-05)	SR_MC *
Tres92	(-6.79E-02, 1.08E-02)	SR_GD
Utas92	(-1.87E-02, 1.79E-03)	TABU_IE *
Utes92	(-2.27E-03, 8.64E-05)	CF_MC
Yorf83	(-8.32E-03, 4.57E-04)	CF_MC
Yue20011	(-9.02E-02, 1.07E-02)	SR_GD
Yue20012	(-7.54E-02, 9.38E-03)	SR_GD
Yue20013	(-2.50E-01, 0.00E+00)	SR_MC *
Yue20021	(-3.45E-02, 4.55E-03)	SR_GD
Yue20022	(-1.26E-02, 9.08E-04)	CF_MC
Yue20023	(-1.52E-02, 2.69E-04)	CF_MC *
Yue20031	(-1.59E-02, 1.65E-03)	CF_MC
Yue20032	(-5.42E-03, 3.68E-04)	CF_MC

### 4.3 Experimental Results

The experimental results of performance comparison of Simple Random, Random Descent, Tabu Search, Choice Function, and Greedy heuristic selection method and all acceptance criteria combinations on 21 different exam timetabling problem instances are statistically evaluated. Each pair has been assigned a ranking. The confidence interval is set to 95% in the  $t$ -test to determine the significant performance variance. Similar to the previous experiments, the combinations that do not have significant performance variances are assigned to the same ranking.

Table 5 gives the average best fitness values for the best performing heuristic selection–acceptance criterion combinations. If several hyper-heuristics share the same ranking, then only one of them appears in the table, marked with \*. Seven combinations that have the top average rankings are presented in Figure 4. According to the results, the Choice Function heuristic selection combined with Monte Carlo acceptance criterion has the best average performance on exam timetabling problems. The hyper-heuristic combinations with acceptance criteria AM and OI do not perform well on any of the problem instances.

**Table 6.** The performance rankings of each heuristic selection–acceptance criterion combination on all problem instances; lower rankings indicate better performance

(a)

H.-h.	Carf92	Cars91	Earf83	Hechs92	Kfus93	Lsef91	Purs93
SR_AM	30.5	26.5	26	26	26	26	26
SR_OI	19.5	19	12.5	16	19	16	8
SR_IE	7.5	7.5	12.5	16	9	11.5	1
SR_MC	15	15	7	7.5	15	11.5	23
SR_GD	7.5	6	8	7.5	1	4.5	9
RD_AM	30.5	31.5	30	31	31	29.5	31.5
RD_OI	19.5	19	20	16	19	20	12.5
RD_IE	7.5	3	12.5	16	9	11.5	4
RD_MC	7.5	11.5	3.5	4.5	9	4.5	20.5
RD_GD	30.5	31.5	30	31	31	29.5	31.5
RP_AM	30.5	31.5	34.5	31	31	34.5	34.5
RP_OI	19.5	19	20	16	19	20	12.5
RP_IE	7.5	3	12.5	16	9	11.5	4
RP_MC	7.5	11.5	3.5	4.5	9	4.5	20.5
RP_GD	30.5	31.5	34.5	31	31	34.5	34.5
RPD_AM	30.5	31.5	30	31	31	29.5	31.5
RPD_OI	19.5	19	20	16	19	20	12.5
RPD_IE	7.5	3	12.5	16	9	11.5	4
RPD_MC	7.5	11.5	3.5	4.5	9	4.5	20.5
RPD_GD	30.5	31.5	30	31	31	29.5	31.5
CF_AM	30.5	26.5	30	31	31	33.5	27
CF_OI	19.5	19	20	16	19	20	12.5
CF_IE	7.5	3	12.5	16	9	11.5	4
CF_MC	7.5	9	1	1.5	3	1	16.5
CF_GD	19.5	19	20	16	19	20	12.5
TABU_AM	30.5	31.5	30	31	31	29.5	28.5
TABU_OI	19.5	19	20	16	19	20	12.5
TABU_IE	7.5	3	12.5	16	9	11.5	4
TABU_MC	7.5	11.5	3.5	4.5	9	4.5	20.5
TABU_GD	30.5	31.5	30	31	31	29.5	28.5
GR_AM	24.5	24.5	24	24.5	24.5	24.5	24.5
GR_OI	19.5	23	20	16	23	20	16.5
GR_IE	7.5	7.5	12.5	16	9	11.5	7
GR_MC	7.5	14	6	1.5	2	4.5	18
GR_GD	24.5	24.5	25	24.5	24.5	24.5	24.5

Table 6 continued (b)

H.-h.	Ryes93	Staf83	Tres92	Utas92	Utes92	Yorf83
SR_AM	26	31	26	26	26	26
SR_OI	19.5	16	19.5	15	16	19.5
SR_IE	8	16	8.5	3.5	16	12
SR_MC	15	4.5	15	19	7	7
SR_GD	8	4.5	1	9	8	8
RD_AM	31	31	31	32.5	31	29.5
RD_OI	19.5	16	19.5	19	16	19.5
RD_IE	8	16	8.5	3.5	16	12
RD_MC	8	4.5	8.5	11.5	4	3.5
RD_GD	31	31	31	32.5	31	29.5
RP_AM	31	31	31	32.5	31	34.5
RP_OI	19.5	16	19.5	19	16	19.5
RP_IE	8	16	8.5	3.5	16	12
RP_MC	8	4.5	8.5	11.5	4	3.5
RP_GD	31	31	31	32.5	31	34.5
RPD_AM	31	31	31	32.5	31	29.5
RPD_OI	19.5	16	19.5	19	16	19.5
RPD_IE	8	16	8.5	3.5	16	12
RPD_MC	8	4.5	8.5	11.5	4	3.5
RPD_GD	31	31	31	32.5	31	29.5
CF_AM	31	26	31	27	31	33
CF_OI	19.5	16	19.5	19	16	19.5
CF_IE	8	16	8.5	3.5	16	12
CF_MC	1	4.5	2	8	1	1
CF_GD	19.5	16	19.5	19	16	19.5
TABU_AM	31	31	31	28.5	31	29.5
TABU_OI	19.5	16	19.5	19	16	19.5
TABU_IE	8	16	8.5	3.5	16	12
TABU_MC	8	4.5	8.5	11.5	4	3.5
TABU_GD	31	31	31	28.5	31	29.5
GR_AM	24.5	24.5	24.5	24.5	24.5	24.5
GR_OI	19.5	16	19.5	23	16	19.5
GR_IE	8	16	8.5	7	16	12
GR_MC	8	4.5	8.5	14	4	6
GR_GD	24.5	24.5	24.5	24.5	24.5	24.5



Table 6 continued (c)

H.-h.	Y011	Y012	Y013	Y021	Y022	Y023	Y031	Y032
SR_AM	26	26	22.5	26	26	9.5	26	28.5
SR_OI	19.5	19.5	31.5	19.5	16	17.5	16	17.5
SR_IE	12	11.5	14	12	12	17.5	16	9
SR_MC	6	11.5	4	8	7.5	3.5	7.5	6.5
SR_GD	1	1	8	1	7.5	7	7.5	8
RD_AM	31	31	22.5	03	29.5	9.5	30	28.5
RD_OI	19.5	19.5	31.5	19.5	20	17.5	16	17.5
RD_IE	12	11.5	14	12	12	17.5	16	17.5
RD_MC	6	5	4	4.5	4	1.5	4	3.5
RD_GD	31	31	22.5	30	29.5	9.5	30	28.5
RP_AM	31	31	22.5	34.5	34.5	34.5	34.5	34.5
RP_OI	19.5	19.5	31.5	19.5	20	28	16	17.5
RP_IE	12	11.5	14	12	12	17.5	16	17.5
RP_MC	6	5	4	4.5	4	25	4	3.5
RP_GD	31	31	22.5	34.5	34.5	34.5	34.5	34.5
RPD_AM	31	31	22.5	30	29.5	31.5	30	28.5
RPD_OI	19.5	19.5	31.5	19.5	20	28	16	17.5
RPD_IE	12	11.5	14	12	12	17.5	16	17.5
RPD_MC	6	5	4	4.5	4	25	4	3.5
RPD_GD	31	31	22.5	30	29.5	31.5	30	32.5
CF_AM	31	31	22.5	30	33	9.5	30	32.5
CF_OI	19.5	19.5	31.5	19.5	20	17.5	16	17.5
CF_IE	12	11.5	14	12	12	17.5	16	17.5
CF_MC	3	5	4	4.5	1	1.5	1	1
CF_GD	19.5	19.5	31.5	19.5	20	17.5	16	17.5
TABU_AM	31	31	22.5	30	29.5	31.5	30	28.5
TABU_OI	19.5	19.5	31.5	19.5	20	28	16	17.5
TABU_IE	12	11.5	14	12	12	17.5	16	17.5
TABU_MC	6	5	4	4.5	4	25	4	3.5
TABU_GD	31	31	22.5	30	29.5	31.5	30	28.5
GR_AM	24.5	24.5	9.5	24.5	24.5	5.5	24.5	17.5
GR_OI	19.5	19.5	31.5	19.5	20	17.5	16	17.5
GR_IE	12	11.5	14	12	12	17.5	16	17.5
GR_MC	2	2	4	4.5	4	3.5	4	6.5
GR_GD	24.5	24.5	9.5	24.5	24.5	5.5	24.5	17.5

## 5 Conclusion

An empirical study on hyper-heuristics is provided in this paper. As an iterative search strategy, a hyper-heuristic is combined with a move acceptance strategy.

Different such pairs are experimented on a set of benchmark functions. According to the outcome, experiments are extended to cover a set of exam timetabling benchmark problem instances.

The experimental results denote that no combination of heuristic selection and move acceptance strategy can dominate over the others on all of the benchmark functions used. Different combinations might perform better on different objective functions. Despite this fact, IE heuristic acceptance criterion yielded better average performance. Considering heuristic selection methods, *Choice Function* yielded a slightly better average performance, but the difference between the performance of *Choice Function* and the other heuristic selection methods were not as significant as it was between acceptance criteria. The experimental results on the exam timetabling benchmark indicated that *Choice Function* heuristic selection method combined with *MC* acceptance criterion performs superior to the rest of the hyper-heuristic combinations.

*Acknowledgement.* This research is funded by TUBITAK (The Scientific and Technological Research Council of Turkey) under grant number 105E027.

## References

1. Ackley, D.: An empirical study of bit vector function optimization. In: Davis, L. (ed.) *Genetic Algorithms and Simulated Annealing*, pp. 170–215. Pitman, London (1987)
2. Ayob, M., Kendall, G.: A Monte Carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In: *InTech 2003. Proceedings of the International Conference on Intelligent Technologies*, Chiang Mai, Thailand, pp. 132–141 (December 2003)
3. Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G.A. (eds.) *Handbook of Metaheuristics*. International Series in Operations Research and Management Science, vol. 57, pp. 457–474. Kluwer, Dordrecht (2003)
4. Burke, E., Newall, J.P., Weare, R.F.: A memetic algorithm for university exam timetabling. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996)
5. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics* 9, 451–470 (2003)
6. Burke, E., Elliman, D., Ford, P., Weare, B.: Examination timetabling in British universities – a survey. In: Burke, E.K., Ross, P. (eds.) *Practice and Theory of Automated Timetabling*. LNCS, vol. 1153, pp. 76–90. Springer, Heidelberg (1996)
7. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaptation of heuristic orderings: models and algorithms for planning and scheduling problems. *Annals of Operations Research* 129, 107–134 (2004)
8. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper heuristic for timetabling problems. *European Journal of Operational Research* 176, 177–192 (2007)
9. Burke, E.K., Petrovic, S., Qu, R.: Case based heuristic selection for timetabling problems. *Journal of Scheduling* 9, 115–132 (2006)

10. Carter, M.W, Laporte, G., Lee, S.T.: Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society* 47, 373–383 (1996)
11. Cowling, P., Kendall, G., Soubeiga, E.: A hyper-heuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
12. Davis, L.: Bit climbing, representational bias, and test suite design. In: *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 18–23 (1991)
13. De Jong, K.: An analysis of the behaviour of a class of genetic adaptive systems. Ph.D. Thesis, University of Michigan (1975)
14. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E., Erben, W. (eds.) *PATAT 2000*. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001)
15. Easom, E.E.: A survey of global optimization techniques. M.Eng. Thesis, University of Louisville, KY (1990)
16. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing* 5, 691–703 (1976)
17. Goldberg, D.E.: Genetic algorithms and Walsh functions: Part I, A gentle introduction. *Complex Systems* 3, 129–152 (1989)
18. Goldberg, D.E.: Genetic algorithms and Walsh functions: Part II, Deception and its analysis. *Complex Systems* 3, 153–171 (1989)
19. Griewangk, A.O.: Generalized descent of global optimization. *Journal of Optimization Theory and Applications* 34, 11–39 (1981)
20. Kendall, G., Mohamad, M.: Channel assignment in cellular communication using a great deluge hyper-heuristic. In: *Proceedings of the 2004 IEEE International Conference on Networks*, pp. 769–773. IEEE Computer Society Press, Los Alamitos (2004)
21. Marin, H.T.: Combinations of GAs and CSP strategies for solving examination timetabling problems. Ph.D. Thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey (1998)
22. Merlot, L.T.G., Boland, N., Hughes, B.D., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) *PATAT 2002*. LNCS, vol. 2740, pp. 207–231. Springer, Heidelberg (2003)
23. Mitchell, M., Forrest, S.: Fitness landscapes: Royal Road functions. In: Baek, T., Fogel, D., Michalewicz, Z. (eds.) *Handbook of Evolutionary Computation*, Institute of Physics Publishing, Bristol and Oxford University Press, Oxford (1997)
24. Özcan, E.: Towards an XML based standard for timetabling problems: TTML. In: *Multidisciplinary Scheduling: Theory and Applications*, vol. 163 (24), Springer, Berlin (2005)
25. Özcan, E., Ersoy, E.: Final exam scheduler – FES. In: *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1356–1363 (2005)
26. Paquete, L.F., Fonseca, C.M.: A study of examination timetabling with multiobjective evolutionary algorithms. In: *MIC 2001. Proceedings of the 4th Metaheuristics International Conference*, pp. 149–154.
27. Petrovic, S., Yang, Y., Dror, M.: Case-based initialisation for examination timetabling. In: *MISTA 2003. Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, Nottingham, pp. 137–154 (August 2003)
28. Rastrigin, L.A.: *Extremal Control Systems. Theoretical Foundations of Engineering Cybernetics Series*. Nauka, Moscow (1974)

29. Rattadilok, P., Gaw, A., Kwan, R.S.K.: Distributed choice function hyperheuristics for timetabling and scheduling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 51–67. Springer, Heidelberg (2005)
30. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley, New York (1981) [translation of Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (1977)]
31. Whitley, D.: Fundamental principles of deception in genetic search. In: Rawlins, G.J.E. (ed.) Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA (1991)
32. Wong, T., Côté, P., Gely, P.: Final exam timetabling: a practical approach. In: Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Winnipeg, vol. 2, pp. 726–731 (May 2002)

# Author Index

- Ahmadi, Samad 383  
Artigues, Christian 67  
Asmuni, Hishammuddin 327  
  
Beyrouthy, Camille 228  
Bilgin, Burak 394  
Burke, Edmund K. 228, 327  
  
de Haan, Peter 267  
Di Gaspero, Luca 40  
Duarte, Alexandre R. 158  
Duran, Guillermo 174  
  
Eley, Michael 364  
  
Ferland, Jacques 53  
Fujiwara, Nobutomo 135  
  
Garibaldi, Jonathan M. 327  
Gendreau, Michel 53, 67  
Gendron, Bernard 53  
  
Haeusler, Edward H. 158  
Hail, Nouredine 53  
Healy, Patrick 119  
Hurkens, Cor 210  
  
Imahori, Shinji 135  
  
Jaumard, Brigitte 53  
  
Kingston, Jeffrey H. 294, 308  
Korkmaz, Emin Erkan 347, 394  
  
Landa-Silva, Dario 228  
Landman, Ronald 267  
Lapierre, Sophie 53  
  
Matsui, Tomomi 135  
McCollum, Barry 3, 228, 327  
  
McMullan, Paul 228  
Meyers, Carol 24  
Miyashiro, Ryuhei 135  
Müller, Tomáš 189  
Murray, Keith 189  
  
Nano, Emilina 105  
Nguyen-Ngoc, Diem-Hang 105  
Noronha, Thiago F. 174  
  
Orlin, James B. 24  
Özcan, Ender 85, 347, 394  
  
Parkes, Andrew J. 228  
Perzina, Radomír 248  
Pesant, Gilles 53  
Post, Gerhard 267  
  
Ranson, David 383  
Ribeiro, Celso C. 147, 158, 174  
Rinaldi, Franca 280  
Rousseau, Louis-Martin 67  
Rudová, Hana 189  
Ruizenaar, Henri 267  
  
Schaerf, Andrea 40  
Serafini, Paolo 280  
Soriano, Patrick 53  
Souyris, Sebastian 174  
  
Ülker, Özgür 347  
Urrutia, Sebastián 147, 158  
  
van den Broek, John 210  
  
Weintraub, Andres 174  
White, Christine A. 105  
White, George M. 105  
Woeginger, Gerhard 210