

Edmund K. Burke, Hana Rudová (Eds.)

Practice and Theory of Automated Timetabling VI

Proceedings of
The 6th International Conference on the
Practice and Theory of Automated Timetabling
30th August – 1st September 2006
Faculty of Informatics, Masaryk University
Brno, The Czech Republic

Volume Editors:

Edmund K. Burke
University of Nottingham
School of Computer Science & Information Technology
Jubilee Campus, Nottingham, NG8 2BB
The United Kingdom of Great Britain and Ireland
E-mail: ekb@cs.nott.ac.uk

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

ISBN 80-210-3726-1

Published by Masaryk University.
Printed from camera-ready submissions by Konvoj, spol. s r.o. (Ltd.)
Copyright holders are authors of the individual submissions.

Preface

On behalf of the Steering Committee and the Programme Committee of the PATAT (Practice and Theory of Automated Timetabling) series of conferences, we would like to welcome you to the sixth conference here in Brno. The PATAT conferences, which are held every two years, bring together researchers and practitioners in all aspects of computer-aided timetable generation. This includes university timetabling, school timetabling, personnel rostering, transportation timetabling, sports scheduling and other areas of the subject. The programme of this year's conference features seventy presentations which represent the state-of-the-art in automated timetabling: there are 4 plenary papers, 17 full papers, 41 extended abstracts, and 8 system demonstrations. We are particularly pleased to welcome system demonstrations which are introduced at PATAT this year for the first time.

As was the case in previous years, a post-conference volume of selected and revised papers is to be published in the Springer Lecture Notes in Computer Science series. This volume will be rigorously refereed by our Programme Committee. The plenary papers and the 17 full papers will automatically go into the reviewing process for this volume but authors will (if they so wish) have the opportunity to revise their papers in the light of feedback from the conference. The authors of the 41 extended abstracts and the 8 system demonstrations are invited to extend their articles and to submit a full paper for publication in this book.

We would like to express our gratitude to the large number of people who have contributed to the organization of the conference. The Steering Committee ensures the ongoing success of the series and the Programme Committee works very hard to referee conference submissions. We are, of course, very grateful to all authors and delegates. We would particularly like to thank the Faculty of Informatics at Masaryk University for hosting the conference. Special thanks 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 and Tomáš Černý for their assistance with type-setting, and to Lenka Bartošková, Dagmar Janoušková, Iva Krejčí and Petra Křivánková for their administrative help. 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. and CELCAT, and the Ministry of Education, Youth and Sports of the Czech Republic for their support under research intent No. 0021622419.

We are very happy to welcome you to Brno. We hope you enjoy your stay here and that you get a chance to explore the city and the surrounding area during your visit. We wish you an informative, useful, and interesting conference. Enjoy it!

July 2006

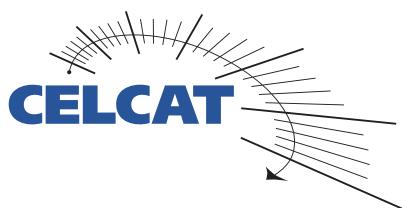
Edmund K. Burke
Hana Rudová

Programme Committee

Edmund K. 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
Amnon Meisels	Ben-Gurion University, Beer-Sheva, Israel
Thiruthallai Nepal	Durban Institute of Technology, South Africa
Jim Newall	eventMAP Ltd., UK
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
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

Conference Sponsors

We gratefully acknowledge the sponsorship of ORTEC bv, eventMAP Ltd. and CELCAT.



Contents

Plenary Presentations	1
Physician Scheduling in Emergency Rooms <i>Michel Gendreau, Jacques Ferland, Bernard Gendron, Noureddine Hail, Brigitte Jaumard, Sophie Lapierre, Gilles Pesant, Patrick Soriano</i>	2
University Timetabling: Bridging the Gap between Research and Practice <i>Barry McCollum</i>	15
Very Large-Scale Neighborhood Search Techniques in Timetabling Problems <i>Carol Meyers, James B. Orlin</i>	36
Measurability and Reproducibility in Timetabling Research: State-of-the-Art and Discussion <i>Andrea Schaerf, Luca Di Gaspero</i>	53
 Full Papers	 63
A Flexible Model and a Hybrid Exact Method for Integrated Employee Timetabling and Production Scheduling <i>Christian Artigues, Michel Gendreau, Louis-Martin Rousseau</i>	64
A Novel Fuzzy Approach to Evaluate the Quality of Examination Timetabling <i>Hishammuddin Asmuni, Edmund K. Burke, Jonathan M. Garibaldi, Barry McCollum</i>	82
The Teaching Space Allocation Problem with Splitting <i>Camille Beyrouty, Edmund K. Burke, J. Dario Landa-Silva, Barry McCollum, Paul McMullan, Andrew J. Parkes</i>	103
An Experimental Study on Hyper-heuristics and Exam Timetabling <i>Burak Bilgin, Ender Özcan, Emin Erkan Korkmaz</i>	123

Timetabling Problems at the TU Eindhoven <i>John van den Broek, Cor Hurkens, Gerhard Woeginger</i>	141
Strategic Employee Scheduling <i>Peter Chan, Michael Hiroux, Georges Weil</i>	157
Ant Algorithms for the Exam Timetabling Problem <i>Michael Eley</i>	167
The KTS High School Timetabling System <i>Jeffrey H. Kingston</i>	181
Hierarchical Timetable Construction <i>Jeffrey H. Kingston</i>	196
An Approach for Automated Surgery Scheduling <i>Karl-Heinz Krempels, Andriy Panchenko</i>	209
Artificial Immune Algorithms for University Timetabling <i>Muhammad Rozi Malim, Ahamad Tajudin Khader, Adli Mustafa</i>	234
An Empirical Investigation on Memes, Self-generation and Nurse Rostering <i>Ender Özcan</i>	246
Solving the University Timetabling Problem with Optimized Enrolment of Students by a Parallel Self-adaptive Genetic Algorithm <i>Radomír Perzina</i>	264
An Extensible Modelling Framework for the Examination Timetabling Problem <i>David Ranson, Samad Ahmadi</i>	281
Generating Personnel Schedules in an Industrial Setting Using a Tabu Search Algorithm <i>Pascal Tellier, George M. White</i>	293
Linear Linkage Encoding in Grouping Problems: Applications on Graph Coloring and Timetabling <i>Özgür Ülker, Ender Özcan, Emin Erkan Korkmaz</i>	303
An Evaluation of Certain Heuristic Optimization Algorithms in Scheduling Medical Doctors and Medical Students <i>Christine A. White, Emilina Nano, Diem-Hang Nguyen-Ngoc, George M. White</i>	320

Extended Abstracts	329
Tackling the University Course Timetabling Problem with an Aggregation Approach <i>Mieke Adriaen, Patrick De Causmaecker, Peter Demeester, Greet Vanden Berghe</i>	330
An Iterative Re-start Variable Neighbourhood Search for the Examination Timetabling Problem <i>Masri Ayob, Edmund K. Burke, Graham Kendall</i>	336
A Simulated Annealing Hyper-heuristic for University Course Timetabling <i>Ruibin Bai, Edmund K. Burke, Graham Kendall, Barry McCollum</i>	345
A Tiling Approach for Fast Implementation of the Traveling Tournament Problem <i>Amotz Bar-Noy, Douglas Moody</i>	351
Understanding the Role of UFOs Within Space Exploitation <i>Camille Beyrouthy, Edmund K. Burke, J. Dario Landa-Silva, Barry McCollum, Paul McMullan, Andrew J. Parkes</i>	359
New Concepts in Neighborhood Search for Permutation Optimization Problems <i>Wojciech Bożejko, Mieczysław Wodecki</i>	363
Scheduling Sport Leagues Using Branch-and-Price <i>Dirk Briskorn</i>	367
Solving Exam Timetabling Problems with the Flex-Deluge Algorithm <i>Edmund K. Burke, Yuri Bykov</i>	370
Examination Timetabling: A New Formulation <i>Edmund K. Burke, Barry McCollum, Paul McMullan, Rong Qu</i>	373
Progress Control in Variable Neighbourhood Search <i>Tim Curtois, Laurens Fijn van Draat, Jan-Kees van Ommeren, Gerhard Post</i>	376
Scheduling with Soft CLP(FD) Solver <i>Tomáš Černý, Hana Rudová</i>	381

Lecture and Tutorial Timetabling at a Tunisian University <i>Abdelaziz Dammak, Abdelkarim Elloumi, Hichem Kamoun</i>	384
An Employee Timetabling Problem in a Maintenance Service of a Software Company <i>Laure-Emmanuelle Drezet, Deborah Chesnes, Odile Bellenguez-Morineau</i>	391
Referee Assignment in Sports Tournaments <i>Alexandre R. Duarte, Celso C. Ribeiro, Sebastián Urrutia</i>	394
Branch-and-cut for a Real-life Highly Constrained Soccer Tournament Scheduling Problem <i>Guillermo Durán, Thiago F. Noronha, Celso C. Ribeiro, Sebastián Souyris, Andrés Weintraub</i>	398
Constructive Algorithms for the Constant Distance Traveling Tournament Problem <i>Nobutomo Fujiwara, Shinji Imahori, Tomomi Matsui, Ryuhei Miyashiro</i>	402
A Study on the Short-Term Prohibition Mechanisms in Tabu Search for Examination Timetabling <i>Luca Di Gaspero, Marco Chiarandini, Andrea Schaerf</i>	406
A Decomposition Approach with Inserted Idle Time Scheduling Subproblems in Group Scheduling <i>Cumhur A. Gelogullari, Rasaratnam Logendran</i>	412
Multi-Site Timetabling <i>Ruben Gonzalez-Rubio</i>	416
Scheduling the Belgian Soccer League <i>Dries Goossens, Frits C. R. Spieksma</i>	420
A Four-phase Approach to a Timetabling Problem in Secondary Schools <i>Peter de Haan, Ronald Landman, Gerhard Post, Henri Ruizenaar</i>	423
Framework for Negotiation in Distributed Nurse Rostering Problems <i>Stefaan Haspeslagh, Patrick De Causmaecker, Greet Vanden Berghe</i>	426

Scheduling Research Grant Proposal Evaluation Meetings <i>Patrick Healy</i>	432
Making Good Rosters for the Security Personnel <i>Han Hoogeveen, Eelko Penninkx</i>	437
Timetabling at German Secondary Schools: Tabu Search versus Constraint Programming <i>Frank Jacobsen, Andreas Bortfeldt, Hermann Gehring</i>	439
A Constructive Heuristic for the Travelling Tournament Problem <i>Graham Kendall, Wim Miserez, Greet Vanden Berghe</i>	443
Computational Complexity Issues in University Interview Timetabling <i>Yuuki Kiyonari, Eiji Miyano, Shuichi Miyazaki</i>	448
Local Search Heuristics for the Teacher/Class Timetabling Problem <i>Yuri Kochetov, Polina Obuhovskaya, Mikhail Paschenko</i>	454
Time Windows and Constraint Boundaries for Public Transport Scheduling <i>Ignacio Laplagne, Raymond S. K. Kwan, Ann S. K. Kwan</i>	458
Minimizing the Carry-Over Effects Value in a Round-Robin Tournament <i>Ryuhei Miyashiro, Tomomi Matsui</i>	460
A Constraint Logic Programming Based Approach to the International Timetabling Competition <i>Patrick Plessas, Mark Wallace, Mauro Bampo</i>	464
Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search <i>Malek Rahoual, Rachid Saad</i>	467
Dynamically Configured λ -optimal Heuristics for Bus Scheduling <i>Prapa Rattadilok, Raymond S. K. Kwan</i>	473
A Dispatching Tool for Railway Transportation <i>Pascal Rebreyend</i>	478
Scheduling the Brazilian Soccer Championship <i>Celso C. Ribeiro, Sebastián Urrutia</i>	481

Scheduling School Meetings <i>Franca Rinaldi, Paolo Serafini</i>	484
Modelling and Solving the Italian Examination Timetabling Problem using Tabu Search <i>Andrea Zampieri, Andrea Schaerf</i>	487
Optimality Aspects with Assigning of Magistrates to Sessions and Teams of the Amsterdam Criminal Court <i>Jan Schreuder</i>	492
Multi-Calendar Appointment Scheduling: Calendar Modeling and Constraint Reasoning <i>Stephanie Spranger, François Bry</i>	496
How to Solve a Timetabling Problem by Negotiation <i>Marie-Hélène Verrons, Philippe Mathieu</i>	502
Experiments with a Form of Double Iterated Search for Use on Hard Combinatorial Problems with Many Objectives <i>Mike B. Wright</i>	506
System Demonstrations	509
Optime: Integrating Research Expertise with Institutional Requirements <i>Edmund K. Burke, Graham Kendall, Barry McCollum, Paul McMullan, Jim Newall</i>	510
Personnel Scheduling in HARMONY <i>Laurens Fijn van Draat, Gerhard Post, Bart Veltman</i>	516
SWOPS (Shift Work Optimized Planning and Scheduling) <i>Dagan Gilat, Ariel Landau, Amnon Ribak, Yossi Shiloach, Segev Wasserkrug</i>	518
Dialog-Based Intelligent Operation Theatre Scheduler <i>Karl-Heinz Krempels, Andriy Panchenko</i>	524
Process Plan Optimization using a Genetic Algorithm <i>Fabian Märki, Manfred Vogel, Martin Fischer</i>	528

THOR: A Tool for School Timetabling <i>Fernando Melício, João P. Caldeira, Agostinho Rosa</i>	532
Automated System for University Timetabling <i>Keith Murray, Tomáš Müller</i>	536
An Integrated Framework for Distributed Timetabling <i>Peter Wilke</i>	542
Author Index	547

Plenary Presentations

Physician Scheduling in Emergency Rooms

Michel Gendreau^{1,2}, Jacques Ferland^{1,2} Bernard Gendron^{1,2}, Noureddine Hail¹, Brigitte Jaumard^{1,3}, Sophie Lapierre^{1,4}, Gilles Pesant^{1,4}, and Patrick Soriano^{1,5}

¹ Interuniversity Centre for Research on Enterprise Networks,
Logistics and Transportation (CIRRELT)

Université de Montréal, C.P. 6128, succ. Centre-ville, Montreal, Canada, H3C 3J7
michelg@crt.umontreal.ca

² Département d'informatique et de recherche opérationnelle, Université de Montréal

³ Concordia Institute for Information Systems Engineering, Concordia University
⁴ Département de mathématiques et génie industriel

École Polytechnique de Montréal

⁵ Service d'enseignement des méthodes quantitatives de gestion, HEC Montréal

Abstract. We discuss the problem of constructing physician schedules in emergency rooms. Starting from practical instances encountered in six 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, 19]. As mentioned by Carter and Lapierre [11], 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 great challenge for them to work in such a place. According to Lloyd *et al.* [23], 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 efforts, 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 there are usually many individual requests and several of them turn out to be conflicting.

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-Cabriñi 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. Finally, we conclude in Section 5.

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 agreement 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 mathematical formulation are not quite different. Indeed, according to Gendreau *et al.* [26], a pure mathematical approach given by Berrada *et al.* in [5, 4, 34] 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 [11] 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 month) 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 [11]:

- *Tangier Emergency Physician Scheduling Software*, by Peake Software laboratories [30];
- *Epsked*, by ByteBloc Medical Software [9];
- *Docs for Windows*, by Acme Express [1];
- *Physician Scheduler 4.0*, by Sana-Med.

These software packages have been sold to emergency departements 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 [8, 21] and some on acyclic rostering [2, 3, 8, 10, 11, 14, 26, 31]. 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[11] 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 [8].

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, 19] 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 [11], 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 8 AM regular shift on Saturday must also cover the 10 AM trauma shift on Sunday; working the 4 PM regular shift on Friday requires working the 4 PM trauma shift on Saturday and the 4 PM 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.

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

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	

4.1 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[14] 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 [12, 25] 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 [22]. 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 ressource constraints on a directed graph [32]. The ressources 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 contraints not explicitely 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 ressources 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 [18], it has been successfully applied to a wide variety of application contexts, such as vehicle routing [16], machine scheduling [28], maximum clique problem [17], quadratic assignement problem [27, 29]. This method has also been applied to the nurse scheduling problem[7, 13], 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 [21] and acyclic schedules [8].

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. to 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 [15] for a comprehensive introduction to the topic).

Buzon's tabu search method for acyclic schedules [8] is in fact an extension and a generalization of previous work by Labb   [21]. 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 2 unbroken weekends. If the schedule associated with this physician in the current solution contains 3 unbroken weekends and 1 broken weekend, then the penalty associated with the weekend constraint would be $(3-2).P_{NBW} + 1.P_{BW}$, 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 [8].

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 [8] 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 [24] 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 [10], Rousseau *et al.* [26], Trilling [31] and Bourdais *et al.* [6]. The work of Rousseau *et al.* [26] 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 [26].

This algorithm was successfully applied to two hospitals: SCH and CNH. The physician scheduling problem that is solved in [26] is formulated as follows:

$$\text{Minimize } f(W)$$

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 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. We have indeed attempted to compare the four approaches described in the previous section and found out 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. <http://www.docs2000.net/productdetailsy2k.asp>, 2000
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** (2000) 139–200
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** (1996) 183–193
6. Bourdais, S., Galinier, P., Pesant, G.: HIBISCUS: A Constraint Programming Application to Staff Scheduling in Health Care. *Principles and Practice of Constraint Programming: Proceedings of the Ninth International Conference (CP'03)*, Kinsale (IE), September 2003, Springer-Verlag Lecture Notes in Computer Science **2833** (2003) 153–167
7. Burke, P., De Causmaecker, P., Vanden Bergue, G.: A hybrid tabu search algorithm for the nurse rostering problem. *Lecture Notes in Computer Science* **1585** (1999) 187–194
8. 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
9. ByteBloc Software. Epsked 3.0 bytebloc software. <http://www.bytebloc.com>, 1995
10. Cangini, G.: A constraint programming local search algorithm for physician scheduling. Centre for Research on Transportation, Université de Montréal, Publication CRT-2000-26, 2000
11. Carter, M.W., Lapierre, S.D.: Scheduling emergency room physicians. *Health Care Management Science* **4** (2001) 347–360
12. Chvátal, V.: *Linear Programming*. Freeman, 1983.
13. Downshall, K.A.: Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operation Research* **106** (1998) 393–407
14. 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
15. Gendreau, M.: An Introduction to Tabu Search. In *Handbook of Metaheuristics*, F.W. Glover and G.A. Kochenberger (eds.), (2003) Kluwer Academic Publishers, Boston, MA, 37–54
16. Gendreau, M., Hertz, A., Laporte, G.: A tabu search algorithm for the vehicle routing problem. *Management Science* **40** (1994) 1276–1290
17. Gendreau, M., Soriano, P., Salvail, L.: Solving the Maximum Clique Problem Using a Tabu Search Approach. *Annals of Operations Research* **41** (1993) 385–403.
18. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* **13** (1986) 533–549
19. Knaunth, P.: Design better shift systems. *Applied Ergonomics* **27** (1996) 39–44
20. Knaunth, P.: The design of shift systems. *Ergonomics* **36** (1993) 15–28
21. Labb  , S.: La confection automatis  e d'horaires pour les m  decins en salles d'urgence. Master's thesis, cole des Hautes tudes Commerciales de Montr  al, Canada, 1998
22. Labit, P.: Am  lioration d'une m  thode de g  n  ration de colonnes pour la confection d'horaire d'infirmi  res. Master's thesis, cole Polytechnique, Montr  al, Canada, 2000
23. Lloyd, S., Shannon, S., Steiner, D.: Burnout, depression, life and job satisfaction among Canadian emergency physicians. *The Journal of Emergency Medicine* **12** (1994) 559–565.
24. Marriott, K., Stuckey, P.J.: *Programming with Constraints: An Introduction*. MIT Press, 1998.
25. Nemhauser, G.L., Wolsey, L.A.: *Integer and Combinatorial Optimization*. Wiley-Interscience, 1998

26. 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 (2000), Paderborn, Germany.
27. Skorin-Kapov, J.: Tabu search applied to the quadratic assignment problem. ORSA Journal on Computing **2** (1990) 33–45
28. Taillard, É.: Some efficient heuristic methods for the flow shop sequencing problem European Journal of Operational Research **47** (1990) 65–74
29. Taillard, É.: Robust taboo search for the quadratic assignment problem Parallel Computing **17** (1991) 443–455.
30. Peake Software laboratories: Tangier emergency physician scheduling software. <http://peakesoftware.com/peake/>
31. Trilling, G.: Génération automatique d'horaires de médecins de garde pour l'hôpital Côte-des-Neiges de Montréal. Centre for Research on Transportation, Université de Montréal, Publication CRT-98-05, 1998.
32. Vovor, T.: Problème de chemins bicritère ou avec contraintes de ressources : algorithmes et applications. Ph.D. thesis, École Polytechnique, Montréal, 1997.
33. Weil, G., Heus, K., Poujade, P., Fran cois, M.: Constraint programming for nurse scheduling. Engineering in medecine and Biology **14** (1995) 417–422
34. Warner, D. M.: Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Programming Approach. Operations Research **24** (1976) 842–856

University Timetabling: Bridging the Gap between Research and Practice

Barry McCollum

School of Computer Science, Queen's University
University Road, Belfast BT7 1NN, N. Ireland
b.mccollum@qub.ac.uk

eventMAP Ltd
21 Stranmillis Road, Belfast BT9 5AF, N. Ireland

Abstract. As an academic in the School of Computer Science at Queen's University, a visiting researcher to the Automated Scheduling, Optimisation and Planning (ASAP) group within the School of Computer Science and IT at the University of Nottingham and Managing Director of eventMAP Limited, a university technological spin out company, the author is in a unique position to provide comments on both the practice and theory of timetabling (automated and otherwise) within the university sector. 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 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

EventMAP Limited was formed in 2002 to exploit the commercial potential of the educational timetabling research carried out by the Automated Scheduling, Optimisation and Planning (ASAP) group at the University of Nottingham and the Knowledge and Data Engineering (KDE) Group within the School of Computer Science at the Queen's University of Belfast. The Company is based in Belfast within the Institute of Electronics, Communications and Information Technology (ECIT) at Queen's University. The Institute, which officially opened in May 2005, represents a new £40M world class centre with a unique focus on blue skies, strategic and

industrial research projects. The Centre brings together internationally renowned research groups specializing in key areas of advanced IT, digital and communications technology. A key feature of the Centre's overall remit is the "spinning out" of industrial based companies exploiting advancements made in research.

The decision to form a company followed identification of the market need for a high quality research led software solution to the scheduling difficulties experienced within the educational sector. The focus of eventMAP Limited is to develop, market and sell examination, course scheduling and space management and planning software into the worldwide higher and further education sector. The preface to the Selected Papers Volume from the Gent PATAT conference [6] stated that "The goal of developing interactive and adaptive systems that build on human expertise and at the same time provide the computational power to reach high-quality solutions continues to be one of the key challenges that currently faces the timetabling research community" This goal is very much shared by eventMAP Limited whose approach is to incorporate knowledge of the extreme complexity of timetabling problems with commercial skills and practical experience with the overall aim of developing and building upon the most recent research in Artificial Intelligence and Operational Research technologies.

The Company aims to develop and implement new practical methodologies and associated algorithmic techniques to enhance the solution of educational timetabling problems across a wide range of real world scenarios. At this early stage of the company's existence, consultancy has been provided and systems implemented in Europe, Australia, New Zealand and America. The fact that work has taken place on a global scale at such an early stage in the company's history is both promising and challenging from a company growth point of view.

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, "*a gap still remains between the output of a successful research project and what is needed for direct use by industry*" [1]. In general, the area of educational timetabling is one such area. The Company 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 current efforts within the Company is to implement software which acts as an enterprise recourse 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 continually updating algorithmic techniques within the software, thus enabling 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 of the main areas which the company is involved with i.e. examination and course timetabling. In each area, a number of challenges are detailed which are based on 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 will help bridge the 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 [2,4,5,6,7], 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 [8,9]. 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 [10]. In addition to the PATAT Conference series, many papers have been published on specific techniques along with reporting of various surveys [11,12]. 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 bench mark data with sufficient room, constraint and solution modelling data. It is expected that the release of six new datasets [13] 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.

From a Company perspective, the latest version of it's flagship examination product, Optime_{xam}, was released in January of this year. An earlier version of the software was presented at the PATAT conference in Konstanz, 2000 [2]. The additional functionality made available through this new version will be discussed at the conference during a software presentation [14]. In general, the aim of improving Optime_{xam} 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 i.e. information and solution modelling.

2.1.1 Information Modelling

Information modelling can be divided into data and constraint modelling. The base examination data from which a workable solution is achieved is 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 approach in relation to educational timetabling [15, 16], 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 [17]. 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 in March of 2006 where students from a Muslim background require Fridays free of examinations [18]. 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 modeling 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 [49] 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 session 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 [21,22] on similarity measurements between datasets, novel techniques need to be investigated to establish how changes in individual data sets from year to year effect the nature of the examination set up and ultimately the algorithmic methods applied.

2.1.2 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 [9], 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 [54, 55] 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 [19] 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 be an improvement phase. In both cases there have been many heuristic techniques applied (see [11]). Recent work has shown promise in relation to using a combination of heuristics in relation to the initial construction [20]. 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 [10, 11]. 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 neighborhood structure has been effective. It is felt that Hyperheuristics approach (heuristics to choose heuristics) [56] 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 [13]. Currently the combinations of algorithmic structures available are Saturation degree (Heuristic Method) [25], Adaptive [26] and Great Deluge during an additional improvement cycle [27]. 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 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 [74].

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 [12,29]. 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 [11]. 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 [30,31,32]. In both cases the timetabling process often involves significant human interaction which, in practice, can turn the process into a room booking exercise [33,34]. Therefore, the construction of a solution is often categorised by finding any timetable that satisfies all of the constraints [12]. 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 surround the entire process and their exists many staff, with differing view points involved. The data required for the process is often difficult to obtain and, as with the examination process, it is often ‘sketchy’ [45,64]. From a staff point of view, fixed views exist on when and where teaching should take place within a predominantly ‘territorialism’ culture [34]. 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 [16,29,31,33,34,35,41,42,45]. Carter [42] stressed the importance of taking into consideration and dealing with the human factors associated with the process of constructing an institutional 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’ data sets 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 [40,63,64,66,67]) 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 advancements have been made with respect to the development of search techniques on bench mark data sets [29,36,41,57,58], there is not much evidence that the work has been translated into actual implementations within a significant number of institutions. Indeed Carter and Laporte [31] 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 modeling 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 [35, 31] 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 addresses 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 [36] 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 and; 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 [36,57] are seen as data which can be used in addition to examination data sets 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 recognized and dealt with. The most realistic formulation of the problem which currently exists can be found at [24]. 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 variance of the related examination timetabling problem [12]. 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 [41]. 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 [64,69,70,71], 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, 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 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 the 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 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 the Company, 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 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. Based on the initial construction and space utilisation, potentially the problem could be reshuffled or indeed amended based on a different measure of optimisation. This optioned is not presently favoured by institutions due to the disruption that would be caused. There are a number of reasons 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 used 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. 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 [64]. Also in [65], 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.

3.2.1 Information Modelling

In terms of information modeling, the main differences with examination timetabling is the much more incomplete nature of the data requirements [45,64] which are much more substantial. Data is 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 a changed solution is close as possible to the initially modeled solution after changes in the original definition. For example see [45].

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, timetable 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 [34]. 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 modeled in terms of constraints and evaluation. In general these usually relate to student and staff preference and space usage.

3.2.2 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 later 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 [51,52,53,68]. 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 maximized 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 [42, 37,31,39,62]. Apart from a few notable exceptions [40], 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 enroll 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 [59], apart from where a lecture event is subdivided. There are also some work dealing with students sectioning problems dated back to 80s [39, 43]. 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 [44] to cluster students in large classes into groups which may later lead to the fewest possible conflicts in timetables. Beyrouthy *et al* [59] 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 [40], meta-heuristics 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 need to be 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 [46]. 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 Math’s school will clash 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 institutional 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 [47]. 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 [61,63]. 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 [38]. 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 [42] e.g. 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 optimization 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 [48,49,50] 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 referencing the literature, detail is provided of the relevant research in both areas. The challenges are presents 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 modeling languages influenced by real world scenarios.
- (iii) Investigation of the role in user interaction in the design of decision support system for course timetabling.

- (vi) Investigation of the need for the reformulation and modeling of the problem. It should be noted 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 of 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.

Acknowledgements

I would like to thank Professor Edmund Burke for his contribution throughout the preparation of this paper.

References

1. EPSRC/ESRC Document Review of Research Status of Operational Research in the UK, 2004.
2. B.McCollum, J.Newall, Introducing Optime: Examination timetabling Software, Proceedings of the 3rd international conference on the Practice and Theory of Automated Timetabling, August 16-18 2000, pp. 485-490, ISBN 3-00-003866-3.
3. E.K.Burke and P.Ross (1996), The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International conference, Edinburgh 1995, Springer Lecture Notes in Computer Science Volume 1153, Springer 1996.
4. E.K.Burke and M.W.Carter (1998), The Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd International conference, Toronto 1997, Springer Lecture Notes in Computer Science Volume 1408, Springer 1998.
5. E.K.Burke and W.Erben (2001), The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International conference, Konstanz 2000, Springer Lecture Notes in Computer Science Volume 2079, Springer 2001.

6. E.K.Burke and P.De Causmaecker, The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th International conference, Gent 2002, Springer Lecture Notes in Computer Science, Volume 2740, Springer 2003.
7. E.K.Burke and M.Trick, The Practice and Theory of Automated Timetabling V: Revised Selected Papers from the 5th International conference, Pittsburgh 2004, Springer Lecture Notes in Computer Science, Volume 3616, Springer 2005.
8. S.Petrovic, V.Patel, Y.Yang, Examination timetabling with fuzzy Constraints, In [7], pp. 313-333.
9. H.Asmuni, E.K.Burke, J.M.Garibaldi, B.McCollum, A Novel Fuzzy Approach to Evaluate the Quality of Examination Timetabling, Accepted for full paper at PATAT06.
10. R.Qu, E.K.Burke, B.McCollum, L.G.T.Merlot and S.Y.Lee, The State of the Art of Examination Timetabling, Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham. To be submitted to Journal of Scheduling.
11. E.K.Burke and S.Petrovic, Recent Research Directions in Automated Timetabling, European Journal of Operational Research, Vol 140 No 2, pp. 266-280.
12. A.Schaerf, A Survey of Automated Timetabling, Artificial Intelligence Review, 13/2, 1999, pp. 87-127.
13. E.K.Burke, B.McCollum, and P McMullan, Examination Timetabling: A New Formulation, Accepted for abstract at PATAT06.
14. E.K.Burke, G.Kendall, B.McCollum, P.McMullan, J.Newall, Optime: Integrating Research Expertise with Institutional Requirements, Accepted for software demonstration at PATAT06.
15. A.Colomni, M.Dorigo and V.Maniezzo, Metaheuristics for High School Timetabling. Computational Optimisation and Applications, Volume 9, pages 275-298, 1998.
16. G.Konstantinow and C.Coakley, Use of Genetic Algorithms in Reactive Scheduling for Course Timetable Adjustments. In [7], pp. 521-522.
17. E.K.Burke, D.G.Elliman, P.H.Ford, and R.F.Weare, Examination Timetabling in British Universities – A Survey. In [3], pp. 76-90.
18. The Times Higher, March 10 2006, page 4.
19. L.Paquete and T.Stützle, Empirical analysis of tabu search for the lexicographic optimisation of the examination timetabling problem. In [6], pp. 413-420.
20. H.Asmuni, E.K.Burke, J.M.Garibaldi and B.McCollum, Fuzzy Multiple Ordering Criteria for Examination Timetabling, Practice and theory of Automated Timetabling V, in [7], pp. 334-353.
21. E. Burke, A.Eckersley, B.McCollum, S.Petrovic, Rong Qu, Identifying Potential Similarity Measures between Exam Timetabling Problem for a Case Based Reasoning system. The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Nottingham, August 2003, pp. 120-136.
22. E.K.Burke, A.J.Eckersley, B.McCollum, S.Petrovic and R.Qu, Using Simulated Annealing to Study Behavior of Various Exam Timetabling Data Sets, 5th Meta-heuristics International Conference MIC03, August 25 - 28 2003, Kyoto International Conference Hall, Kyoto, Japan.
23. E.K. Burke, P.De Causmaecker, S.Petrovic, G.Vanden Berghe, Metaheuristics for handling Time Interval Coverage Constraints in Nurse Scheduling. Applied Artificial Intelligence, Vol. 20, No. 3.
24. <http://www.diegm.uniud.it/satt/projects/EduTT/>
25. E.K.Burke, J.Newall and R.F.Weare. A Simple Heuristically Guided Search for the Timetable Problem, Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (EIS'98), E. Alpaydin, C Fyté (eds), University of La Laguna, Spain 1998, pp. 574-579, published by ICSC Academic Press.
26. E.K.Burke and J.P.Newall, Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings, Annals of Operations Research 129, 2004, pp. 107-134.

27. E.K.Burke, J.Newall, Enhancing Timetable Solutions with Local Search Methods. Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT 2002), In [6], pp. 195-206.
28. E.K.Burke, G.Kendall, B.McCollum, P.McMullan and J.Newall, A preference based measurement of optimization, Internal ASAP Technical Report eMAP/2006/02a.
29. O.Rossi-Doria, M.Samples, M.Birattari, M.Chiarandini, M.Dorigo, M.Gambardella, J.Knowles, M.Manfrin, M.Mastrolilli, B.Paechter, L.Paquete, T.Stutzle, A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem, In [6], pp. 329-351.
30. S.Petrovic, and E.K.Burke, Educational Timetabling, in Joseph Leung (Ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman & Hall/CRC Press, 2004, pp. 45-1 - 45-23.
31. M.W.Carter, and G.Laporte, Recent Developments in Practical Course Timetabling, In [4], pages 3-19.
32. V.A.Bardadym, Computer Aided School and Timetabling: The New Wave, In [3], pp. 22-45.
33. B.McCollum, P.McMullan, J.Newall, JP.Lane, A workable scheduling algorithm, The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Nottingham, August 2003, pp. 570-572.
34. B.McCollum, The Implementation of a Centrally computerised timetabling system in a large British Civic University, In [4], pp. 237-254.
35. B.McCollum, Bridging the gap between research and practice: University timetabling in the real world – KEYNOTE, Proceedings of the 47th Annual Operational Society Conference (OR47), September 2005, Chester, UK.
36. <http://www.metaheuristics.org>
37. E.K.Burke, J.H.Kingston, D. de Werra, Applications to Timetabling. The Handbook of Graph Theory, Gross J., Yellen J. (eds.). Chapman Hall/CRC Press, pp. 445-474, 2004.
38. B.McCollum, 2003-2004 Academic Timetabling: Analysis of Staff and Student perception. Internal report eMAP04/02/01.
39. G.Laporte and S.Desroches, The Problem of Assigning Students to Course Section in a Large Engineering School. Computers and Operations Research, 13, pp. 387-394, 1986.
40. D.W.Corne, J.Kingston, Addressing the Availability-Based Laboratory/Tutorial Timetabling Problem with Heuristics and Metaheuristics. Proceedings of the 4th International Conference on the Practice and Theory of Automated timetabling pp. 136-140.
41. S.Abdullah, E.K.Burke, B.McCollum, An Investigation of Variable Neighbourhood Search for the Course Timetabling Problem, The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA05), pp. 413-427 New York, July 18-21, 2005.
42. M.W Carter, A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo, in [5], pp. 64 - 84.
43. J.Aubin and J.A.Ferland, A Large Scale Timetabling Problem. Computers and Operations Research, 16: pp. 67-77, 1989.
44. M.Amintoosi, J.Haddadina, Feature Selection in a Fuzzy Student Sectioning Algorithm, in [7], pp. 147-160.
45. T.Muller, H.Rudova, R.Bartak, Minimal Perturbation Problem in Course timetabling, In [7], pp. 126-146.
46. J.Thompson and K.Dowsland. A Robust Simulated Annealing Based Examination Timetabling System. Computers Operations Research, Volume 25, pp. 637-648, 1998.
47. B.McCollum, M.McKillop, P. McMullan, Course Scheduling: The Division of Lecture Events into Tutorials. Internal Report CSS/04/12abs.

48. E.K. Burke and J.D Landa Silva, The influence of the Fitness Evaluation Method on the Performance of Multiobjective Optimisers, accepted for publication in the European Journal of Operational Research, 2004.
49. E.K. Burke, P.De Causmaecker, S.Petrovic, G.Vanden Berghe, A Multi Criteria Meta-heuristic Approach to Nurse Rostering, Proceedings of Congress on Evolutionary Computation, CEC2001, IEEE Press, 2001, pp. 1139-1146.
50. Deb, Pratap, Agarwal, Meyarivan, (2002), A Fast and Elitist Multi-objective Genetic Algorithm, IEEE Trans. Evol. Comp. 6(2), pp. 182-197.
51. E.K. Burke, E.Kingston, J.Pepper, A standard data format for timetabling instances. In [4], pp. 213-223.
52. A.Chand, A Constraint Based Generic Model for Representing Complete University Timetabling Data, Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pp. 125-150.
53. L.Reis, E.Oliveira, A Language for Specifying Complete Timetabling Problems. Practice and Theory of Automated Timetabling III, In [5], pp. 322-341.
54. E.K.Burke, Y.Bykov and S.Petrovic, A Multi-Criteria Approach to Examination Timetabling, In [6] pp. 118-131.
55. S.Petrovic, Y.Bykov, A Multiobjective Optimisation Technique for Exam Timetabling Based on Trajectories. In [6], pp. 179-92, Aug 21-23, 2002.
56. EK.Burke, B.McCollum, A.Meisels, S.Petrovic, and R.Qu, A Graph-Based Hyper Heuristic for Educational Timetabling Problems. Accepted for publication in the European Journal of Operational Research, 2006.
57. K.Socha, J.Knowles, M.Samples, A Max-Min Ant System for the University Course Timetabling Problem. Proceedings of the 3rd International Workshop on Ant Algorithms, ANTS 2002, Lecture Notes in Computer Science 2463 (10), pp. 1-13.
58. S.Abdullah, E.K.Burke, B.McCollum, Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for Course Timetabling. In Proceedings of MIC 05: The 6th Meta-Heuristic International Conference, Vienna, Austria, 22-26 Aug 2005.
59. C.Beyrouty, E.K.Burke, J.Landa-Silva, B.McCollum, P.McMullan, A.J.Parkes, The Teaching Space Allocation Problem with Splitting. Accepted as a paper for PATAT06.
60. C.Beyrouty, E.K.Burke, J.Landa-Silva, B.McCollum, P.McMullan, A.J.Parkes Understanding the Role of UFOs Within Space Exploitation. Accepted as an abstract for PATAT06.
61. 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.
62. M.Amintoosi, J. Haddadnia, Feature Selection in a Fuzzy student Sectioning Algorithm, In [7], pp. 147-160.
63. S.Geller, Timetabling at the University of Sheffield, UK-hardening the incremental approach to timetable development, Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pp. 499-500.
64. R.G.Rubio, D.P. Munoz, A Timetable Production System Architecture for Course and Exams, Proceedings of the 5th International Conference on the Practice and Theory of Automated timetabling, pp. 567-570.
65. J.H. Kingston and B.Yin-Sun Kynn, A Software Architecture for Timetable Construction, In [6], pp. 342-350.
66. E.Ozan, A.Alkan, Timetabling using a Steady State Genetic Algorithm, Proceedings of the 4th International Conference on the Practice and Theory of Automated timetabling, pp. 104-106.

67. A.Cumming, B.Paechter, R.C.Rankin, Post-Publication Timetabling, Proceedings of the 3rd International Conference on the Practice and Theory of Automated timetabling, pp. 107-108.
68. L.P.Reis, E.Oliveira, A Language for Specifying Complete Timetable Problems, in [6], pp. 322-341.
69. M.Dimopoulou, P.Miliotis, Implementing a University Course and Examination Timetabling System in a Distributed Environment, Proceedings of the 3rd International Conference on the Practice and Theory of Automated timetabling, pp. 148-151.
70. T. Muller, R. Barak, Interactive Timetabling: Concepts, Techniques and Practical Results, Proceedings of the 4th International Conference on the Practice and Theory of Automated timetabling, pp. 58-72.
71. A.Piechowiak, J.Ma, R.Mandiau, An Open Interactive timetabling Tool, In [7], pp. 34-50.
72. L.T.G.Merlot, N.Borland, B.D.Hughes, P.J. Stuckey, A Hybrid Algorithm for the Examination Timetabling Problem, In [6], pp. 207-231.
73. E.K.Burke, J.P.Newall and R.F.Weare, A Memetic Algorithm for University Exam Timetabling, In [3], pp. 241-250.
74. E.K. Burke, G.Kendall, J.Newall, E.Hart, P.Ross and S.Schulenburg, Hyper-Heuristics: An Emerging Direction in Modern Search Technology, Chapter 16 in Handbook of Meta-Heuristics, (eds. F. Glover and G. Kochenberger), pp. 457-474, Kluwer, 2003.

Very Large-Scale Neighborhood Search Techniques in Timetabling Problems

Carol Meyers¹ and James B. Orlin²

¹ Operations Research Center, Massachusetts Institute of Technology
carol@mit.edu *

² Sloan School of Management, Massachusetts Institute of Technology
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. We conclude by encouraging the timetabling community to further examine the promising potential of these techniques in practice.

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 *examination timetabling*, in which a set of exams is to be scheduled over a set of time periods, and *course timetabling*, where a set of courses must be scheduled over the length of an entire semester.

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

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 *automated* timetabling algorithms.

Recent surveys on automated timetabling (see [21, 23, 24, 43]) illustrate the wide array of methods that have been applied to timetabling problems. Traditional techniques tested in timetabling include *direct heuristics* [34], which fill up the timetable one event at a time and resolve conflicts by swapping exams, and a reduction to the *graph coloring* problem [38], where events are associated with vertices of a graph and edges with potential conflicts. More modern heuristics include *memetic* [20] and *genetic algorithms* [19, 27, 30], which use techniques inspired by evolutionary biology; *simulated annealing* algorithms [18, 46], where nonimproving solutions are permitted with progressively decreasing probability; *tabu search* heuristics [26, 42], where a list of recently visited timetables are forbidden to be visited; and *constraint logic programming* approaches [25], 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, 13, 33] (see Section 2). We believe there are many untapped possibilities for useful algorithms in this context.

1.2 Very Large-Scale Neighborhood Search

Neighborhood search algorithms (also known as *local 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 *very large* 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 *relatively efficient* amount of time. Such algorithms tend to search *implicitly* 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 [5]. The first of these is *variable depth* methods, which partially search an exponentially large neighborhood by using heuristics. The second kind are *network flow-based* methods, which use network flow techniques to search over the neighborhood and identify improving neighbors. The third main category consists of neighborhoods based on *restrictions* of NP-hard problems that are solvable in polynomial time. Ahuja, Ergun, Orlin, and Punnen [5, 6] 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 [28, 35, 39], the quadratic assignment problem [7], vehicle routing problems [2, 29], the capacitated minimum spanning tree problem [10], the generalized assignment problem [50, 51], and parallel machine scheduling problems [3]. 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 *combined* within the framework of other heuristic methods, such as tabu search [32, 33] and scatter search [41], to provide further computational improvements. See [5, 6] 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 *examination* 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 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 *optimized crossover* 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 *both* 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 *functional annealing* 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 *partitioning* 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, Ahmadi, Burke, and Dror [1].

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

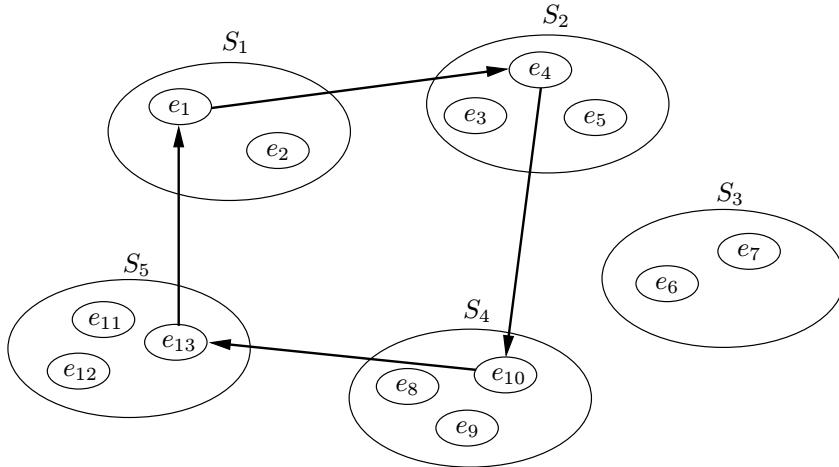


Fig. 1. The cyclic exchange neighborhood.

2-opt operation have been studied previously in the timetabling community by Alvarez-Valdez, Martin, and Tamarit [12], Colomni, Dorigo, and Maniezzo [26], and Schaerf [42], among others. If instead we do not require exam e_{i_k} to move into set S_1 , then we call the operation a *path exchange*, which can be described by the path of exams $e_{i_1} - e_{i_2} - e_{i_3} - \dots - e_{i_k}$. We can show mathematically that path exchanges may be modeled as a special case of cyclic exchanges, by adding dummy nodes as appropriate [10].

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 [49] and Ahuja, Orlin, and Sharma [9, 10] 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 [48] and further examined by Thompson and Psaraftis [49]. Rather than explicitly searching over each possible solution in the neighborhood, the improvement graph allows us to *implicitly* search the neighborhood for improving solutions. This helps dramatically 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 *subset-disjoint* 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 [48] 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 *same cost*.

This result suggests that to effectively search the cyclic exchange neighborhood, we need only to identify *negative cost subset-disjoint cycles* in the improvement graph. Unfortunately, although the problem of finding a general negative cost cycle is solvable in polynomial time [8], the problem of finding a negative cost subset-disjoint cycle is NP-hard [47, 48]. However, Thompson and Psaraftis [49] and Ahuja, Orlin, and Sharma [10] 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, Orlin, and Sharma'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, Orlin, and Sharma, improving solutions in the neighborhood can be found successfully in practice. This suggests that the cyclic exchange neighborhood is 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, Ahmadi, Burke, and Dror [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, Orlin, and Sharma [10]. They additionally introduced an exponential Monte Carlo acceptance criterion (see [14]) 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.

Jha [33] 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-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 *swap* operation, since it consists of swapping the time slots of a pair of exams. The *2-opt neighborhood* is defined as the set of all possible solutions that can be reached from a given solution by performing a single 2-opt move.

Many papers in the timetabling literature have used neighborhood search over the 2-opt neighborhood to refine timetabling solutions, though not necessarily using that name and most often in conjunction with other techniques. Alvarez-Valdes, Martin, and Tamarit [12] used 2-opt moves combined with tabu search in finding solutions for timetabling problems in the Spanish school system. Schaerf [42] combined tabu search and the randomized nonascendent method with 2-opt neighborhood search techniques in solving high school timetabling problems. Colorni, Dorigo, and Maniezzo [26] used 2-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 [22] 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-opt moves.

It should be noted that while 2-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 [37]. In applying genetic algorithms to timetabling 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 $\{T_1^{t+1}, T_2^{t+1}, \dots, T_K^{t+1}\}$ is generated from the population at time t according to the *fitness* of each of the candidate solutions T_i^t , along with *crossover* and *mutation* operations.

The *fitness* 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, K individuals of 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_ℓ^k represents the time period in which exam e_ℓ is scheduled in timetable T_k .

The crossover operation can take several forms, of which the *fixed point* 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 ℓ 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 *two-point crossover*, where two random positions ℓ_1 and ℓ_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 *multi-point crossovers* 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 *mutation* operation is used to ensure diversity of the timetables generated. In this operation, a given position ℓ in timetable T_k is selected with some (small) probability P_m , and exam e_ℓ is reassigned from period p_ℓ^k in which it is currently scheduled to another randomly selected time period. This has the effect of ‘mutating’ the ℓ 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 *randomly*, and the resulting child is created *without regard to the objective function*. Hence occasionally the fitness of a child can deviate quite widely from the fitness of its parents. Aggarwal, Orlin, and Tai [4] suggested instead choosing the *best* child from all possible children, building on an idea of Balas and Niehaus [15] 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 *best* 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 *pair* 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 *best* 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, Orlin, and Tai [4] 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 [15]. 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 [16].

Ahuja, Orlin, and Tiwari [11] 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 [40] 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, Paixão, and Portugal [36] 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 *explicitly* in timetable scheduling problems is unfortunately NP-hard, via a transformation from the MINIMUM SET COVER problem (see [31]). 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, we believe the strong results obtained for the crossover method in other problems (see [4, 11]) 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 [44, 45], 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, [18] and [46]), 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 *annealing function*, 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 [45]. 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 *volatility* 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.

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. 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 [17]).

```

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.

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 [44, 45] 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 *randomized first improvement* 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 *not worse* than that of simulated annealing, and for large numbers of iterations the exiting probability is about $|N(x)|$ times *greater* 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 *best improvement* 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 [44, 45] give a thorough computational study of functional annealing algorithms applied to the quadratic assignment problem. They show that the functional annealing algorithm performs *significantly 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 *statistical learn-*

ing technique 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 *very large-scale 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 *pure* 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 [44, 45] 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 *much the same way* that simulated annealing algorithms are currently used. (See [18] and [46] 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 [43] 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 time-

tabling problems. The applications range from one that has been used before in timetabling problems (the *cyclic exchange neighborhood*), to one that has been widely used in contexts other than timetabling (*optimized crossover* in genetic algorithms), to a relatively new concept that we believe has a great potential for timetabling problems (*functional annealing* algorithms).

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.

References

1. S. Abdullah, S. Ahmadi, E. Burke, and M. Dror. Applying Ahuja-Orlin's large neighborhood for constructing examination timetabling solution. In *Proceedings of the Fifth International Conference on the Practice and Theory of Automated Timetabling*, number 3616 in Lecture Notes in Computer Science, pages 413–420, Pittsburgh, PA, 2004. Springer.
2. R. Agarwal, R. Ahuja, G. Laporte, and Z. Shen. A composite very large-scale neighborhood search algorithm for the vehicle routing problem. In *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, chapter 49. Chapman & Hall/CRC, Boca Raton, FL, 2003.
3. R. Agarwal, Ö. Ergun, J. Orlin, and C. Potts. Solving parallel machine scheduling problems with variable depth local search. Working Paper, Operations Research Center, MIT, Cambridge, MA, 2004.
4. C. Aggarwal, J. Orlin, and R. Tai. Optimized crossover for the independent set problem. *Operations Research*, 45:226–234, 1997.
5. R. Ahuja, Ö. Ergun, J. Orlin, and A. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.
6. R. Ahuja, Ö. Ergun, J. Orlin, and A. Punnen. Very large-scale neighborhood search: theory, algorithms, and applications. Working Paper, Operations Research Center, MIT, Cambridge, MA, 2006.
7. R. Ahuja, K. Jha, J. Orlin, and D. Sharma. Very large-scale neighborhood search for the quadratic assignment problem. Working Paper, Operations Research Center, MIT, Cambridge, MA, 2002.
8. R. Ahuja, J. Orlin, and T. Magnanti. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, 1993.
9. R. Ahuja, J. Orlin, and D. Sharma. Very large-scale neighborhood search. *International Transactions in Operational Research*, 7:301–317, 2000.
10. R. Ahuja, J. Orlin, and D. Sharma. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001.
11. R. Ahuja, J. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27:917–934, 2000.

12. R. Alvarez-Valdes, G. Martin, and J. Tamarit. Constructing good solutions for the Spanish school timetabling problem. *Journal of the Operational Research Society*, 47:1203–1215, 1996.
13. P. Avella, B. D'Auria, S. Salerno, and I. Vasil'ev. Computational experience with very large-scale neighborhood search for high-school timetabling. Working Paper, Research Center on Software Technology, Università del Sannio, Italy, 2006.
14. M. Ayob and G. Kendall. A Monte Carlo hyper heuristic to optimise component placement sequencing for multi-head placement machine. In *Proceedings of the Fourth International Conference on Intelligent Technologies*, pages 132–141, Chiang Mai, Thailand, 2003. Institute for Science and Technology Research and Development, Chiang Mai University.
15. E. Balas and W. Niehaus. Finding large cliques in arbitrary graphs by bipartite matching. In *Clique, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, pages 29–53. AMS, 1996.
16. E. Balas and W. Niehaus. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, 4:107–122, 1998.
17. D. Bertsimas and J. Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, Belmont, MA, 1997.
18. B. Bullnheimer. An examination scheduling model to maximize students' study time. In *Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling*, number 1408 in Lecture Notes in Computer Science, pages 78–91, Toronto, Canada, 1998. Springer.
19. E. Burke, D. Elliman, and R. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 605–610, Pittsburgh, PA, 1995. Morgan Kaufmann.
20. E. Burke, J. Newall, and R. Weare. A memetic algorithm for university exam timetabling. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling*, number 1153 in Lecture Notes in Computer Science, pages 241–250, Edinburgh, United Kingdom, 1996. Springer.
21. E. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140:266–280, 2002.
22. M. Carter. A comprehensive course timetabling and student scheduling system at the University of Waterloo. In *Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling*, number 2079 in Lecture Notes in Computer Science, pages 64–82, Konstanz, Germany, 2000. Springer.
23. M. Carter and G. Laporte. Recent developments in practical examination timetabling. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling*, number 1153 in Lecture Notes in Computer Science, pages 3–21, Edinburgh, United Kingdom, 1996. Springer.
24. M. Carter and G. Laporte. Recent developments in practical course timetabling. In *Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling*, number 1408 in Lecture Notes in Computer Science, pages 3–19, Toronto, Canada, 1998. Springer.
25. C. Cheng, L. Kang, N. Leung, and G. White. Investigations of a constraint logic programming approach to university timetabling. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling*, number 1153 in Lecture Notes in Computer Science, pages 112–129, Edinburgh, United Kingdom, 1996. Springer.
26. A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high-school timetabling. *Computational Optimization and Applications*, 9:275–298, 1998.

27. D. Corne, P. Ross, and H. Fang. Fast practical evolutionary timetabling. In *Proceedings of the Artificial Intelligence and Simulation of Behavior Workshop on Evolutionary Computing*, number 865 in Lecture Notes in Computer Science, pages 251–263, Leeds, United Kingdom, 1994. Springer.
28. V. Deineko and G. Woeginger. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical Programming*, 87:255–279, 2000.
29. Ö. Ergun. *New neighborhood search algorithms based on exponentially large neighborhoods*. PhD dissertation, Massachusetts Institute of Technology, June 2001.
30. H. Fang. *Genetic Algorithms in Timetabling and Scheduling*. PhD dissertation, University of Edinburgh, September 1994.
31. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, NY, 1979.
32. F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65:223–253, 1996.
33. K. Jha. *Very large-scale neighborhood search heuristics for combinatorial optimization problems*. PhD dissertation, University of Florida, June 2004.
34. W. Junginger. Timetabling in Germany - a survey. *Interfaces*, 16:66–74, 1986.
35. S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
36. H. Lourenço, J. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35:331–343, 2001.
37. M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.
38. G. Neufeld and J. Tartar. Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM*, 17:450–453, 1974.
39. A. Punnen and S. Kabadi. Domination analysis of some heuristics for the traveling salesman problem. *Discrete Applied Mathematics*, 119:117–128, 2002.
40. C. Ribeiro and D. Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path relinking. In *Proceedings of the Second Brazilian Workshop on Bioinformatics*, pages 97–102, Macaé, Brazil, 2003. SBC.
41. S. Ropke and D. Pisinger. 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.
42. A. Schaefer. 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.
43. A. Schaefer. A survey of automated timetabling. *Artificial Intelligence Review*, 13:87–127, 1999.
44. D. Sharma and S. Sukhapesna. Functional Annealing technique for very large-scale neighborhood search. In Preparation, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 2006.
45. S. Sukhapesna. *Generalized annealing algorithms for discrete optimization problems*. PhD dissertation, University of Michigan, September 2005.
46. J. Thompson and K. Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25:637–648, 1998.
47. P. Thompson. *Local search algorithms for vehicle routing and other combinatorial problems*. PhD dissertation, Massachusetts Institute of Technology, May 1988.
48. P. Thompson and J. Orlin. The theory of cyclic transfers. Working Paper OR 200-89, Operations Research Center, MIT, Cambridge, MA, 1989.

49. P. Thompson and H. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
50. M. Yagiura and T. Ibaraki. Recent metaheuristic algorithms for the generalized assignment problem. In *Proceedings of the Twelfth International Conference on Informatics Research for Development of Knowledge Society Infrastructure*, pages 229–237, Kyoto, Japan, 2004. IEEE Computer Society.
51. M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover. A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. *Discrete Optimization*, 1:87–98, 2004.

Measurability and Reproducibility in Timetabling Research: State-of-the-Art and Discussion

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

email: schaerf@uniud.it l.digaspero@uniud.it

Abstract In this paper, we first illustrate the state-of-the-art in timetabling research w.r.t. 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 left out other equally-important timetabling problems, such as for example 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, more than in the past. 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 in 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 neither the *measurability* nor the *reproducibility* of the results. The reader is thus "left alone" to judge the quality of the paper, and to understand what can be learnt 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 and fundamental paper [14]. Nevertheless, we believe that this is particularly true in timetabling research, probably because of its shorter tradition 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, aside 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 state-of-the-art 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 left 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 details, 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 for it.

2 State of the art

In this section, we review the most remarkable 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 and statistical tools. Finally, we discuss the issue of the objective validation of the proposed results.

2.1 Problem Formulations & 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 [23].

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 cooperate for the solution. Furthermore, algorithms developed for more complex *ad hoc* formulations, can be tested on the basic standard ones so as to asses their objective quality.

For the EXAMINATION TIMETABBING problem (ETTP), Carter *et al* [7] provide a set of formulations which differ to each other based on some components

of the objective function. They also provided a set of benchmark instances [6] extracted from real data. Formulations and benchmarks by Carter have stimulated a large body of research, so that many researches (see, e.g., [4,12,8]) 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 way. At present, all available instances and the corresponding best results (up to 2003) are published on the Web [17].

We call 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. On the contrary, 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 [22] and it has been the subject of the timetabling competition TTComp2002¹ [21]. The formulation proposed for TTComp2002 has also become quite standard, and many researchers have used it for their work (see, e.g., [16,9]). Twenty artificial instances were generated for the competition, and they are available from the TTComp2002 web page. In addition, a few other have been proposed (and made available via web) in [24].

As mentioned above, the COURSE TIMETABLING problem (CTTP), consists in the weekly scheduling of the lectures of a set of university course. Unfortunately, no standard formulation has emerged from the community for CTTP so far. Up to our knowledge, the only formulation available on the Web [11] together with a set of instances is the one proposed by ourselves in [13], along with 4 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 in the community of a formulation 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 TTComp2002 the input data comes in a single file containing the scalar values (events, rooms, room features, 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 web page and paper.

¹ In the competition the problem is named CTTP, where C stands for course; 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

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 to be maintained and checked. For example, it happened that Carter’s ETTP instances were replicated incorrectly on other web sites. This was due to the presence of a few newlines added in the files, that led to different (less constrained) instances. This unfortunate episode, that might have caused wrong results in some papers, 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. A few structured format have been proposed in the literature, such as STTL [5,15] and TTML [19]. In [20], the authors go even beyond the language, proposing a multi-layer architecture for the specification and the management of timetabling problems. Up to our knowledge, however, these proposals have received a quite limited attention so far. 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 & 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 one it is necessary to specify not only the instances used, but also on which features they are compared (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 on non-determinism in the solution process.

For the TTComp2002, 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, averaged upon the 20 proposed instances. Unfeasible solutions where 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 the so-called *Mongolian horde* approach: run as many trials as you can with no time limit and report only the best of all of them.

Up to our knowledge, the TTComp2002 has been the sole attempt in this respect. All other comparison 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 has been claimed (and published) in 2002, 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 TTComp2002, 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, unfortunately, no validation tool is available. Validation is currently based only on voluntary peer-to-peer interaction based on exchanges of solutions and values.

For our formulation of the CTTP, we have developed a web page [11] that allows the other researchers to download the problem formulation, the data format, and the benchmark instances. More importantly, everybody is allowed also to validate his/her own solutions, and to insert it among the results obtained for that 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 [14], although we try to extract the advices by Johnson that we believe best suit 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 methods for obtaining a sound comparison of the different techniques. 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 their competitors.

However, as it is common expertise of 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., [1,27]). 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 plenty of methods for correctly coping with several situations that arises in practice (see, e.g., [18]).

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 tool for the R statistical package [3]. This 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.

It is worth to notice that the statistical comparison of algorithms outlined in this section is based on the assumption of having access at previous results (or better at 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 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 work that would probably be too time-consuming for a researcher, aside 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 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 (changing names for privacy reasons, if necessary) in the study, 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 authors².

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. [10,25], or a research area [26]. These web sites normally exhibit references to papers, people, problem formulations, 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 systems*). A PMS is a web application (rather than a web site) that should allow the users to interact with the application performing automatically all the following tasks:

Add results: New results are first validated, and then possibly inserted in the database along with 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 events.

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, ...

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 in different formats so that coherent data can be proposed in different format to the community.

Organize on-line competitions: Competitions on specific instances and with registered participants and fixed deadlines can be organized 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.

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 malicious uses, and therefore some of the actions mentioned

² Many researchers (including ourselves!) experienced the frustration of loosing their solutions (or other data) for some of the problems they worked on.

above would need the approval of the administrator before becoming effective. This however would be just a Yes/No button, so that the administrator is pushed to answer shortly.

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.

References

1. R. Barr, B. Golden, J. Kelly, M. Resende, and W. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1:9–32, 1995.
2. Mauro Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Belgium, 2004.
3. Mauro Birattari. *The RACE pacakge*, April 2005.
4. E. Burke and J. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.
5. E. Burke, P. Pepper, and J. Kingston. A standard data format for timetabling instances. In E. Burke and M. Carter, editors, *Proc. of the 2nd Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-97), selected papers*, volume 1408 of *Lecture Notes in Computer Science*, pages 213–222, Berlin-Heidelberg, 1997. Springer-Verlag.
6. M. W. Carter. Carter’s test data. URL: <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/>, 2005. Viewed: June 1, 2006, Updated: June 7, 2005.
7. M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 74:373–383, 1996.
8. S. Casey and J. Thompson. Grasping the examination scheduling problem. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002), selected papers*, volume 2740 of *Lecture Notes in Computer Science*, pages 232–244, Berlin-Heidelberg, 2003. Springer-Verlag.
9. M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid approach for the university course timetabling problem, 2004. Accepted for publication on the Journal of Scheduling.
10. Joseph Culberson. Graph coloring page. URL: <http://www.cs.ualberta.ca/~joe/Coloring/>, 2006. Viewed: June 1, 2006, Updated: March 31, 2004.
11. Luca Di Gaspero, Alessandro Fontanel, and Andrea Schaerf. Educational timetabling uniud. URL: <http://www.diegm.uniud.it/satt/projects/EduTT/>, 2006. Viewed: June 1, 2006, Updated: May 30, 2006.
12. Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In E. Burke and W. Erben, editors, *Proc. of the 3rd Int. Conf. on*

- the Practice and Theory of Automated Timetabling (PATAT-2000), selected papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer-Verlag, Berlin-Heidelberg, 2001.
13. Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002), selected papers*, volume 2740 of *Lecture Notes in Computer Science*, pages 262–275, Berlin-Heidelberg, 2003. Springer-Verlag.
 14. D. S. Johnson. A theoretician’s guide to the experimental analysis of algorithms. In M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society, 2002. Available from <http://www.research.att.com/~dsj/papers.html>.
 15. Jeffrey H. Kingston. Modelling timetabling problems with STTL. In E. Burke and W. Erben, editors, *Proc. of the 3rd Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2000), selected papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 309–321. Springer-Verlag, Berlin-Heidelberg, 2001.
 16. Philipp Kostuch. The university course timetabling problem with a three-phase approach. In Edmund Burke and Michael Trick, editors, *Proc. of the 5th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2004), selected papers*, volume 3616 of *Lecture Notes in Computer Science*, pages 109–125, Berlin-Heidelberg, 2005. Springer-Verlag.
 17. Liam Merlot. Public exam timetabling data sets. URL: <http://www.or.ms.unimelb.edu.au/timetabling>, 2005. Viewed: June 1, 2006, Updated: October 13, 2003.
 18. D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, sixth edition, 2005.
 19. E. Özcan. Towards an XML-based standard for timetabling problems: TTML. In G. Kendall, E. Burke, S. Petrovic, and M. Gendreau, editors, *Proc. of the 1st Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA-03), selected papers*, pages 163–185. Springer, 2005.
 20. Y. Lu P. D. Causmaecker, P. Demeyer and G. Vanden. Using web standards for timetabling. In *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002)*, pages 238–257, 2003.
 21. Ben Paechter, Luca Maria Gambardella, and Olivia Rossi-Doria. International timetabling competition, web page. URL: <http://www.idsia.ch/Files/ttcomp2002/>, 2003. Viewed: June 1, 2006, Updated: July 10, 2003.
 22. Olivia Rossi-Doria et al. A comparison of the performance of different metaheuristic on the timetabling problem. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002), selected papers*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351, Berlin-Heidelberg, 2003. Springer-Verlag.
 23. Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
 24. Krzysztof Socha, Joshua Knowles, and Michael Sampels. A MAX-MIN ant system for the university timetabling problem. In *Proc. of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 1–13, Brussels, Belgium, 2002. Springer-Verlag, Berlin, Germany. Data available from: <http://iridia.ulb.ac.be/~msampels/tt.data/>.

25. Michael Trick. Challenge traveling tournament instances, web page. URL: <http://mat.gsia.cmu.edu/TOURN/>, 2005. Viewed: June 1, 2006, Updated: May 18, 2006.
26. The web site of the EURO working group on automated timetabling (watt). URL: <http://www.asap.cs.nott.ac.uk/watt/>, 2002. Viewed: June 1, 2006, Updated: November 28, 2005.
27. Jiefeng Xu, Steve Chiu, and Fred Glover. Fine-tuning a tabu search algorithm with statistical tests. *Int. Transactions on Operational Research*, 5(3):233–244, 1998.

Full Papers

A flexible model and a hybrid exact method for integrated employee timetabling and production scheduling

Christian Artigues^{1,2} and Michel Gendreau², Louis-Martin Rousseau²

¹ LIA – Université d’Avignon

339 chemin meinajariés, Agroparc, BP 1228, 84911 Avignon Cedex 9, FRANCE

christian.artigues@univ-avignon.fr

² CRT – Université de Montréal

C.P. 6128, succursale Centre-ville, Montréal, QC H3C 3J7 CANADA

{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 at 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 [1–7] 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 [8–14]. 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 24h basis) whereas the employee timetabling has to be discretized in periods (on a 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 [8–14] among others.

We focus hereafter on some recent papers presenting different models and solution methods. Cordeau *et al.* [11] 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 3-phase method, adding progressively the integrity constraints. More recently, Mercier *et al.* [14] have improved the robustness of the proposed model. Their method reverses the benders decomposition proposed in [11] by considering the crew scheduling problem as the master problem.

Haase and Fridberg [10] 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 [15] 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.* [13] 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 an 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.* [16] 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. Danniels and Mazzola [1] 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.* [3] propose the same approach in a parallel machine context. Bailey *et al.* [2] and Alfares and Bailey [4] 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 lies in a maximal number of workers per period. In [6], 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 [17] 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.* [7]. They extend the model proposed in [1] 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 [18], 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 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 a two machines provided by the authors show the computation of this duration of a job amounts to solve 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 reduced to the assignment of an employee to each machine, the duration of the jobs depending of the employee performance.

Drezet and Billaut [19] 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 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 [20, 21]. In this paper we focus on only one of the problems presented in [22] 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 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 [23]. 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 problem

The model of integration proposed by [18] 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 (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 (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 (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 an employee 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 constraints 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 to take 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 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 machine 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.

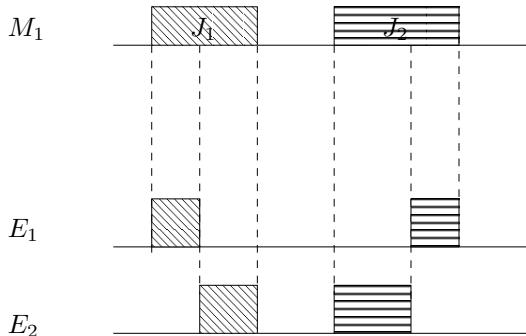


Fig. 1. a 1-machine and 2-employee example

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 Θ denote the time horizon for the timetabling problem. Furthermore, we assume 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 θ , $0 \leq \theta < \Theta$.

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

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

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

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

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

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

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

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

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

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

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

Such constraints allow 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 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 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 1 hour and the timetabling period is 4 hours, then an aggregated information of the activities to perform during each 4 hours period has to be provided. In this purpose, values $R_{jat\theta}$ need not be integers if the activity a generated by job j in timetabling period θ occupies only a portion of an employee's work capacity. In figure 3, each job is assumed to require 0.25 employee per time unit and generate a single activity. Then, the demand for employees able to perform this activity is displayed for each time table period.

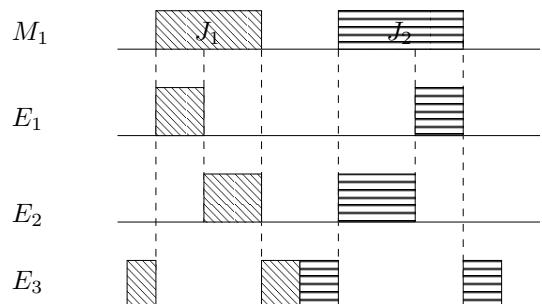


Fig. 2. a 1-machine and 3-employee example

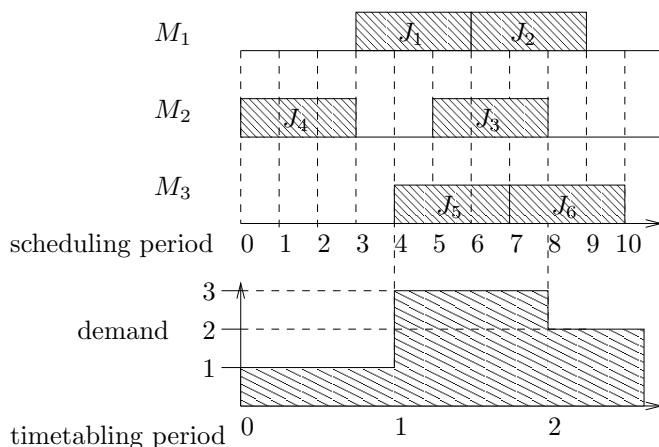


Fig. 3. a 1-machine and 2-employee example

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 θ 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 θ , binary value $y_{sa\theta}$ is such that $y_{sa\theta} = 1$ if the schedule performs activity a at time θ and $y_{sa\theta} = 0$ otherwise. C_s denote the cost of a schedule $s \in \mathcal{S}_e$. 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):

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

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

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

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

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

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

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

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

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

4 A Constraint Programming model

Constraint programming formulations have been proposed for production scheduling [24] and for employee timetabling [22]. 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 θ and a demand variable $\delta_{\theta a} \in \mathbb{N}$ giving the number of employees required for activity a during period θ . Consider the following constraint satisfaction problem (CSP):

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

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

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

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

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

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 δ through generic constraint ϕ 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 θ , $\delta_{\theta e}$ variables must have value a in the activity assignment vector $(a_{\theta e})_{e \in \mathcal{E}}$ of employees during period θ . Last, constraints (44) express the employee specific and regulation constraints through the global regular language membership constraints **regular** [25], restricting the sequence of values taken by the assignment variables to belong to the regular language associated to Π .

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 [22], 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_{empl} denote the total satisfaction cost of employees. The considered problem can be denoted

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

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

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

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

$$(40) \dots (44)$$

Constraints (46) enforce the makespan value. Constraint (47) defines the total cost C_{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_{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_{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 ϕ (42) have to ensure that once a complete assignment of the start time variables is computed, the demand variables δ 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 [26] for a precise description of those algorithms.

For domain reduction of the demand and activity variables δ 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 θ 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 ϕ . 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_{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, $\underline{C}_{\text{empl}} + \tilde{C}_{ea\theta} > Z$, 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 6 jobs and 4 machines, comprising 24 operations. We consider a set of 15 employees and a set of 4+1 activities. The job operations processing times vary from 1 to 10. We assume one time unit corresponds to one hour. We define a timetabling period as a 8-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 2 production activities out of 4. Each break must be of at least 2 consecutive shifts (16-hour break). There is a 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 ϕ 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 4 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 1. Column Inst gives the instance number. 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⁻) and CPU(H⁻) give the same values for the hybrid method used without reduced cost-based filtering.

Inst	Mks*	cost(M)	CPU(M)	#fails(M)	cost*	CPU(H)	#fails(H)	CPU(H ⁻)	#fails(H ⁻)
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

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

For the proposed instances, the makespan minimisation problem is very easy since CP always solves the problem in less than 0.2s. 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 methods 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.

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 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 behaviour 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. Daniels, R.L., Mazolla, J.B.: Flow shop scheduling with resource flexibility. *Operations Research* **42**(3) (1994) 504–522
2. Bailey, J., Alfares, H., Lin, W.: Optimization and heuristic models to integrate project task and manpower scheduling. *Computers and Industrial Engineering* **29** (1995) 473–476
3. Daniels, R.L., Hoopes, B.J., Mazolla, J.B.: Scheduling parallel manufacturing cells with resource flexibility. *Management science* **42** (1996) 1260–1276
4. Alfares, H., Bailey, J.: Integrated project task and manpower scheduling. *IIE Transactions* **29**(9) (1997) 711–717
5. Chen, Z.: Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research* **129** (2004) 135–153
6. 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** (2004) 121–129
7. Daniels, R.L., Mazolla, J.B., Shi, D.: Flow shop scheduling with partial resource flexibility. *Management Science* **50**(5) (2004) 658–669
8. 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**(2) (1983) 63–211
9. 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*. Kluwer (1998) 57–93
10. Haase, K., Friberg, C.: An exact algorithm for the vehicle and crew scheduling problem. In Wilson, N., ed.: *Computer-Aided Transit Scheduling*. Volume 471 of *Lecture Notes in Economics and Mathematical Systems*. Springer (1999) 63–80

11. Cordeau, J.F., Stojković, G., Soumis, F., Desrosiers, J.: Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation science* **35** (2001) 375–388
12. Klabjan, D., Johnson, E., Nemhauser, G.: Airline crew scheduling with time windows and plane count constraints. *Transportation science* **36** (2002) 337–348
13. Freling, R., Huisman, D., Wagelmanss, A.: Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling* **6**(1) (2003) 63–85
14. 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** (2005) 1451–1476
15. Haase, K., Desaulniers, G., Desrosiers, J.: Simultaneaous vehicle and crew scheduling in urban mass transit systems. *Transportation science* **35**(3) (2001) 286–303
16. 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** (1996) 285–302
17. Faaland, B., Schmitt, T.: Cost-based scheduling of workers and equipment in a fabrication and assembly shop. *Operations Research* **41**(2) (1993) 253–268
18. Haït, A., Baptiste, P., Brauner, N., Finke, G.: Approches intégrées à court terme (2005) chapter 6 in [27].
19. Drezet, L.E., Billaut, J.C.: Tabu search algorithms for a predictive and a reactive project scheduling problem. In: 6th Metaheuristics International Conference, Vienna, Austria (2005)
20. Ernst, A., Jiang, H., Krishnamoorthy, M., SierK, D.: Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* **153** (2004) 3–27
21. Soumis, F., Pesant, G., Rousseau, L.M.: Gestion des horaires et affectation du personnel (2005) chapter 4 in [27].
22. Demassey, S., Pesant, G., Rousseau, L.M.: Constraint programming based column generation for employee timetabling. In: 7th International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'05), Prague (2005)
23. Pinedo, M.L.: Planning and Scheduling in Manufacturing and Services. Springer Series in Operations Research and Financial Engineering. Springer (2005)
24. Baptiste, P., Pape, C.L.: Disjunctive constraints for manufacturing scheduling: Principles and extensions. *International Journal of Computer Integrated Manufacturing* **9**(4) (1996) 306–310
25. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Tenth International Conference on Principles and Practice of Constraint Programming -CP'04. Volume 3258 of Lecture Notes in Computer Science., Springer-Verlag (2004) 482–495
26. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling. Kluwer (2001)
27. Baptiste, P., Giard, V., Haït, A., Soumis, F., eds.: Gestion de production et ressources humaines. Presses Internationales Polytechnique (2005)

A Novel Fuzzy Approach to Evaluate the Quality of Examination Timetabling

Hishammuddin Asmuni¹, Edmund K. Burke¹, Jonathan M. Garibaldi¹, and Barry McCollum²

¹ 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

² 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.

Keywords - Timetabling, fuzzy sets, multi-criteria decision making

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 in any circumstances. Any timetable solution that satisfies all the hard constraints specified is considered as 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 [7, 10, 17, 25], Tabu Search [8, 18, 20, 26], Simulated Annealing [24], Constraint Programming [1, 4, 19], Case Based Reasoning [11, 27] and Fuzzy Methodologies [2, 3, 22, 27] have been successfully applied to timetabling problems.

In 1996, Carter *et al.* [14] introduced a set of examination timetabling benchmark data. This benchmark data set consists of 13 problem instances. Originally these data came from real university examination timetabling problems. Therefore, it was expected that these data sets varied considerably in terms of resources given/availability, constraints specified and how the quality of the constructed timetable were evaluated. For the sake of generality, these data sets were then simplified such that only the following constraints were considered:

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 slot, and others. This is discussed in more detail in the following section.

Previous studies such as [3] and [22], demonstrated that fuzzy reasoning is a promising technique that can be used both for modeling 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 specific objectives specified earlier, the method is general in the sense that a user could, in principle, define additional criteria he or she wished 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 the timetabling officer considering 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 Function

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 [14]. 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. that 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 [14] 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 are 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 (adapted from 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.* [10] 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 required 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.* [22] 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. The final value does not, for example, 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), with extensive experience of real-world timetabling, having spent 12 years as a timetabling officer and with continuing links with the timetabling industry, has expressed (via private communication) 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 optimised for 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 combination. 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 importance 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, in that they can implement both linear and non-linear combinations. The nature of fuzzy systems that allows the use of linguistic terms to express the systems' behaviours provides a transparent representation of the nonlinear system under

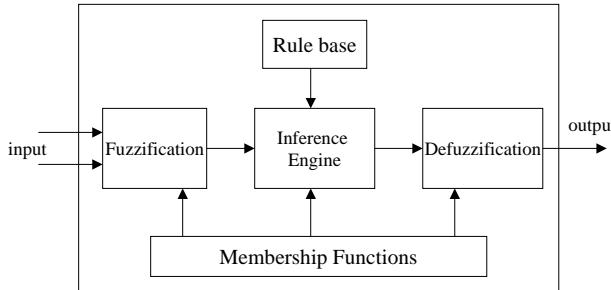


Fig. 1. Components of a fuzzy system

consideration. 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', which 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 subsection 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. Figure 1 shows the 5 interconnected components of a fuzzy system. The fuzzification component computes the membership grade for each crisp input variables based on the membership functions defined. The 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. The 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 [28]. 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 [21], 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 Cox [16] for a simple treatment or Zimmerman [29] 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. Beside 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 the simplest possible to adequately represent the problem. Gaussian functions of the form $e^{-(x-c)^2/\sigma^2}$, where c and σ are constants, are used to define the fuzzy set for each linguistic term. This is on the basis that they are the simplest and

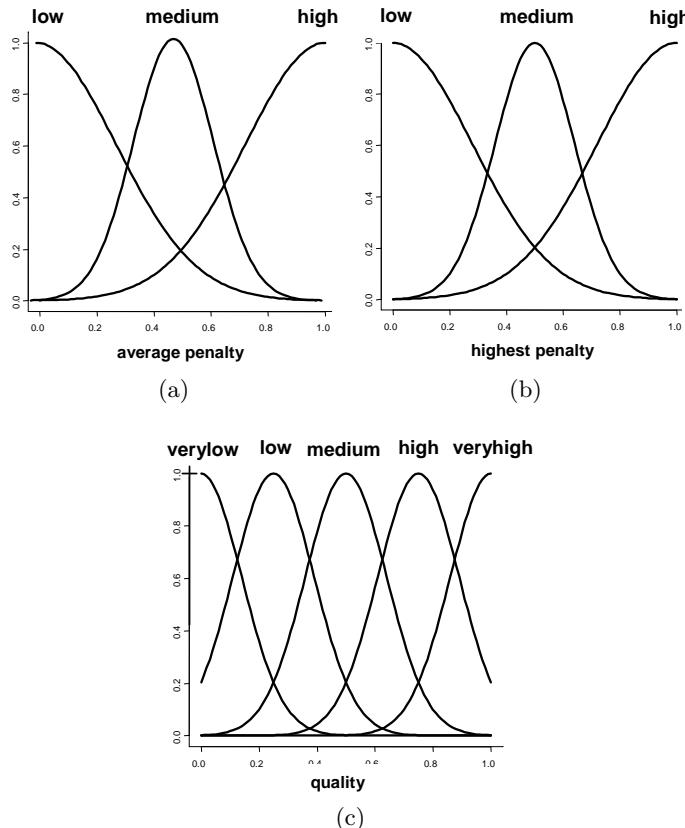


Fig. 2. Membership functions for input and output variables

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 Figure 2 (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 more fair timetable for all students. Based upon this knowledge we defined a fuzzy rule set consisting of all 9 possible combinations. Each rule set connects the input variables to a single output variable, *quality*. The fuzzy rule set is presented in Figure 3. As stated above, standard Mamdani style fuzzy inference was used to obtain the fuzzy output for a given set of

- 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. 3. Fuzzy rules for *Fuzzy Evaluation System*

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.

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-83 data set (from Carter *et al.*— see below for full details of the data sets used) an average penalty of 160.42 was obtained, whereas for UTA-S-92, the average penalty was 3.57.

As can be seen in Figure 2(a) and Figure 2(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:

$$v' = \frac{(v - lowerBound)}{(upperBound - lowerBound)}$$

where v is the actual value in the initial range [$lowerBound, upperBound$]. In effect, the range [$lowerBound, upperBound$] 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 Carter *et al.*'s [14] benchmark data sets were used. The 12 instances in these benchmark data sets, with different characteristics and various level of complexity, are shown in Table 1.

Table 1. Examination timetabling problem characteristics

Data Set	Number of slots (T)	Number of exams (N)	Number of students (S)
CAR-F-92	32	543	18419
CAR-S-91	35	682	16925
EAR-F-83	24	190	1125
HEC-S-92	18	81	2823
KFU-S-93	20	461	5349
LSE-F-91	18	381	2726
RYE-F-92	23	486	11483
STA-F-83	13	139	611
TRE-S-92	23	261	4360
UTA-S-92	35	622	21266
UTE-S-92	10	184	2750
YOR-F-83	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 8 different heuristics to construct the timetable solutions, for each of which the algorithm was run 5 times to obtain a range of solutions. However, due to the nature of the heuristics used, in some case, a few of the constructed timetable solutions have the same proximity cost value. Therefore, for the purpose of standardization, 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, the 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 complexity (see Table 1), the determination of the initial range for each data set is not a straight-forward 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-83* data set is around 130 [15]. 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*) [23].

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 4(a) and 4(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 function has performed as desired, in that the overall (fuzzy) quality of the solutions varies from close to zero to close to one.

Table 2. Minimum and maximum values for *Average Penalty* and *Highest Penalty* obtained from the 35 timetable solutions for each data set

Data Set	<i>Average Penalty</i>		<i>Highest Penalty</i>	
	Minimum Value	Maximum Value	Minimum Value	Maximum Value
CAR-F-92	4.54	11.42	65.0	132.0
CAR-S-91	5.29	13.33	68.0	164.0
EAR-F-83	37.02	71.28	105.0	198.0
HEC-S-92	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-83	160.75	194.53	227.0	284.0
TRE-S-92	8.67	17.25	68.0	129.0
UTA-S-92	3.57	8.79	63.0	129.0
UTE-S-92	28.07	56.34	83.0	129.0
YOR-F-83	39.80	64.48	228.0	331.0

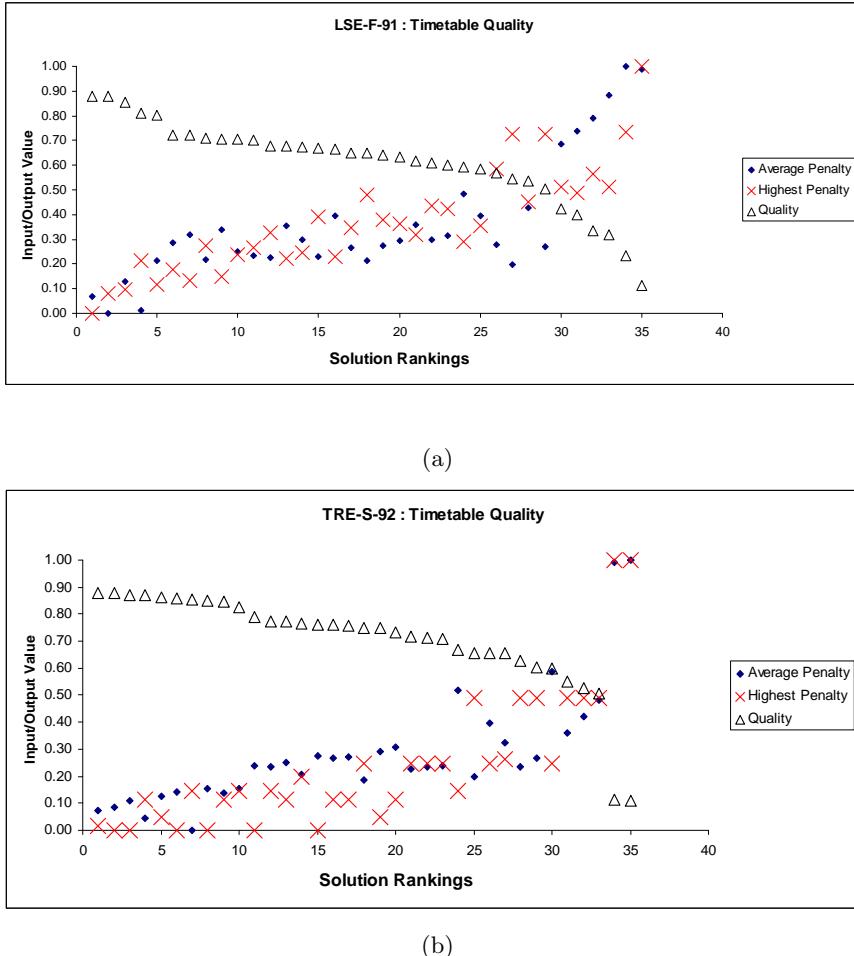


Fig. 4. Indicative illustrations of the range of normalised inputs and associated output obtained for the *LSE-F-91* and *TRE-S-92* data sets

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*]’) indicates 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 the data sets are presented, based on the ranking produced when the boundary range [*minValue*, *maxValue*] was used.

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-92	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-91	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-83	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-92	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
LSE-F-91	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
	21	160.746	227	0.888536	1	0.215426	1
	20	161.151	227	0.887829	2	0.214107	2
STA-F-83	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
	19*	9.311	69	0.880078	1	0.478231	2
	8*	9.389	68	0.878204	2	0.479078	1
TRE-S-92	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
	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
UTA-S-92	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
	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
UTE-S-92	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
	21	39.801	234	0.883004	1	0.372139	1
	8*	44.158	233	0.837983	2	0.363036	3
YOR-F-83	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

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 Figures 4(a) and 4(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’ solution. Regardless, in terms of functionality, these results indicate 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-92* (in Table 3) and *UTA-S-92* 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-83* (in Table 4), the ranking change is quite marked.

Overall, the performance of the fuzzy evaluation system utilizing 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-83*, 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 2(c)).

Table 5. Range of timetable quality

Data Set	Range $[0, maxValue]$		Range $[minValue, maxValue]$	
	Worst	Best	Worst	Best
	Solution	Solution	Solution	Solution
CAR-F-92	0.111464	0.534427	0.111464	0.888503
CAR-S-91	0.111464	0.557585	0.111464	0.888524
EAR-F-83	0.111465	0.467867	0.111465	0.868135
HEC-S-92	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-83	0.111464	0.215426	0.111464	0.888536
TRE-S-92	0.111476	0.479078	0.111488	0.880078
UTA-S-92	0.111464	0.532771	0.111464	0.888536
UTE-S-92	0.111464	0.438284	0.111464	0.879116
YOR-F-83	0.120046	0.372139	0.213388	0.883004

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 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 5(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 5(b) illustrates the situation when the normalised input values fall in the regions of linguistic term that corresponding 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 2(c)). As 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-92* data set are presented in Figure 4. As can be seen, the input values (Figures (b) and (c)) are concentrated in the middle regions (0.4 – 0.7) of the graphs when the

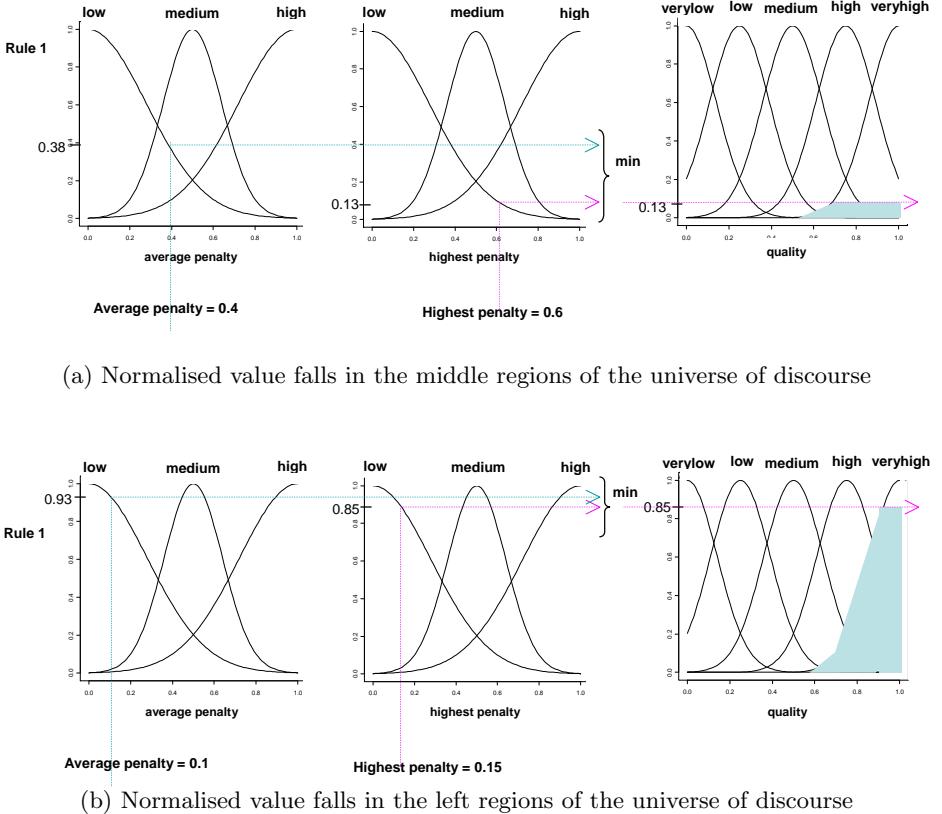
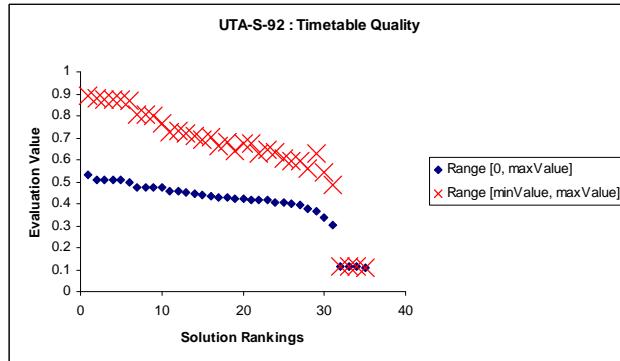
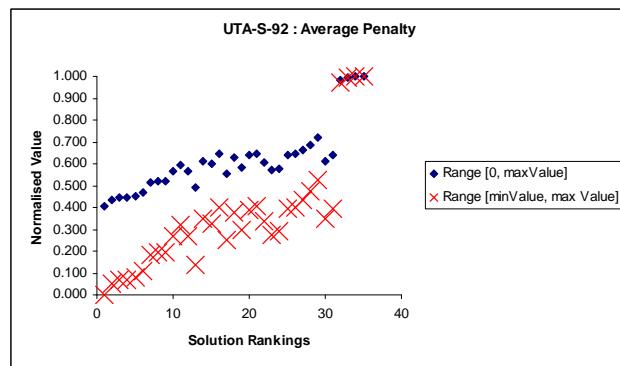


Fig. 5. Firing level for **Rule 1** with different normalised input values

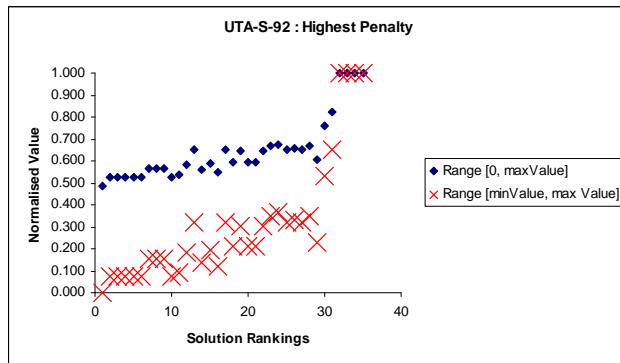
boundary range $[0.0, maxValue]$ was used. In contrast, when the boundary range $[minValue, maxValue]$ was used, the input values were concentrated in the bottom regions of the graphs. Based upon the defined fuzzy rules, we know that the timetable quality increases with a decrease in both input values. Indeed, this behavior of the output can be observed for both boundary setting (see Figure 4(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 are 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



(a)



(b)



(c)

Fig. 6. A graphical comparison of the effect of the two boundary settings for UTA-S-92

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.

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

References

1. S. Abdennadher and M. Marte. University Course Timetabling Using Constraint Handling Rules. *Journal of Applied Artificial Intelligence*, 14(4):311–326, 2000.
2. H. Asmuni, E. K. Burke, and J. M. Garibaldi. A Comparison of Fuzzy and Non-Fuzzy Ordering Heuristics for Examination Timetabling. In A. Lotfi, editor, *Proceeding of 5th International Conference on Recent Advances in Soft Computing 2004*, pages 288–293, 2004.
3. H. Asmuni, E. K. Burke, J. M. Garibaldi, and Barry McCollum. Fuzzy Multiple Heuristic Orderings for Examination Timetabling. In Burke and Trick [13], pages 334–353.
4. P. Boizumault, Y. Delon, and L. Peridy. Constraint Logic Programming for Examination Timetabling. *The Journal of Logic Programming*, 26(2):217–233, 1996.
5. E. K. Burke, Y. Bykov, J. Newall, and S. Petrovic. A Time-Predefined Local Search Approach to Exam Timetabling Problems. *IIE Transactions*, 36(6):509–528, June 2004.
6. E. K. Burke, D. G. Elliman, P. H. Ford, and R. F. Weare. Examination Timetabling in British Universities - A Survey. In Burke and Ross [12], pages 76–90.

7. E. K. Burke, D. G. Elliman, and R. F. Weare. A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA'95, Pittsburgh, USA, 15th-19th July 1995)*, pages 605–610, San Francisco, CA, USA, 1995. Morgan Kaufmann.
8. E. K. Burke, G. Kendall, and E. Soubeiga. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6):451–470, Dec 2003.
9. E. K. Burke and J. P. Newall. A Multistage Evolutionary Algorithm for the Timetable Problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.
10. E. K. Burke, J. P. Newall, and R. F. Weare. A Memetic Algorithm for University Exam Timetabling. In Burke and Ross [12], pages 241–250.
11. E. K. Burke, S. Petrovic, and R. Qu. Case Based Heuristic Selection for Timetabling Problems. *Journal of Scheduling*, 9(2):99–113, 2006.
12. E. K. Burke and P. Ross, editors. *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, volume 1153 of *Lecture Notes in Computer Science*. Springer, 1996.
13. E. K. Burke and M. A. Trick, editors. *Practice and Theory of Automated Timetabling V, 5th International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18-20, 2004, Revised Selected Papers*, volume 3616 of *Lecture Notes in Computer Science*. Springer, 2005.
14. M. W. Carter, G. G. Laporte, and S. Y. Lee. Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the Operational Research Society*, 47:373–383, 1996.
15. S. Casey and J. Thompson. GRASPing the Examination Scheduling Problem. In E. K. Burke and P. D. Causmaecker, editors, *Practice and Theory of Automated Timetabling IV (PATAT 2002, Gent Belgium, August, selected papers)*, volume 2740 of *Lecture Notes in Computer Science*, pages 232–244, Berlin Heidelberg New York, 2003. Springer-Verlag.
16. E. Cox and M. O'Hagen. *The Fuzzy Systems Handbook : A Practitioner's Guide to Building, Using and Maintaining Fuzzy Systems*. AP Professional, Cambridge, MA, 1998.
17. S. Deris, S. Omatsu, H. Ohta, and P. Saad. Incorporating Constraint Propagation in Genetic Algorithm for University Timetabling Planning. *Engineering Applications of Artificial Intelligence*, 12:241–253, 1999.
18. L. Di Gaspero and A. Schaerf. Tabu Search Techniques for Examination Timetabling. In E.K. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III (PATAT 2000, Konstanz Germany, August, selected papers)*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117, Berlin Heidelberg New York, 2001. Springer-Verlag.
19. C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building University Timetables Using Constraint Logic Programming. In Burke and Ross [12], pages 130–145.
20. G. Kendall and N. Mohd Hussin. A Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at the MARA University of Technology. In Burke and Trick [13], pages 270–293.
21. Mamdani, E. H., and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.
22. S. Petrovic, V. Patel, and Y. Yang. University Timetabling With Fuzzy Constraints. In Burke and Trick [13].

23. 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.
24. J. M. Thompson and K. A. Dowsland. A Robust Simulated Annealing Based Examination Timetabling System. *Computers & Operations Research*, 25(7/8):637–648, 1998.
25. H. Ueda, D. Ouchi, K. Takahashi, and T. Miyahara. Comparisons of Genetic Algorithms for Timetabling Problems. *Systems and Computers in Japan*, 35(7):1–12, 2004. Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol.J86-D-I, No. 9, September 2003, pp. 691-701.
26. G. M. White, B. S. Xie, and S. Zonjic. Using Tabu Search with Longer-Term Memory and Relaxation to Create Examination Timetables. *European Journal of Operational Research*, 153:80–91, 2004.
27. Y. Yang and S. Petrovic. A Novel Similarity Measure for Heuristic Selection in Examination Timetabling. In Burke and Trick [13].
28. L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.
29. H. J. Zimmerman. *Fuzzy Set Theory and Its Applications*. Kluwer Academic Publishers, 3rd edition, 1996.

The Teaching Space Allocation Problem with Splitting

Camille Beyrouthy^{1*}, Edmund K. Burke¹, J. Dario Landa-Silva¹, Barry McCollum^{2,3}, Paul McMullan^{2,3}, and Andrew J. Parkes¹

¹ School of Computer Science & IT,
University of Nottingham, Nottingham NG8 1BB, UK.

{cbb,ekb,jds,ajp}@cs.nott.ac.uk

² Queen's University of Belfast, Belfast, BT7 1NN, UK
{b.mccollum,p.p.mcmullan}@qub.ac.uk

³ Realtime Solutions Ltd, 21 Stranmillis Road, Belfast, BT9 5AF
b.mccollum@realtimesolutions-uk.com

Abstract. A standard problem within universities is that of Teaching Space Allocation; 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 the course corresponds to seminars or tutorials, and although hundreds of students are enrolled, each individual class, or event, should be just tens of students in order to meet student and institutional preferences.

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-off surfaces. The explorations themselves are based on a local search method we introduce that attempts to optimise the space utilisation by means of a “dynamic splitting” strategy. The local moves are designed to improve utilisation and the satisfaction of 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:

* Contact Author.

- **space management:** near-term planning
- **space planning:** long-term planning, including capacity planning

A fundamental stage of capacity planning is 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”; 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, we need to build in excess capacity [14, 15].

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 unsplittable “events” (or “courses” or “classes”). Meaning, that they are “atomic”, 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.

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 other 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, and this greatly reduces the chances of obtaining a conflict-free timetable. Splitting such a course into multiple sections can greatly help 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 (e.g. [17, 4, 7, 8, 10, 9, 16, 11]) 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 optimising the satisfaction of soft constraints such as the avoidance of unpopular times. The best-known problem that consists of “timetabling with splitting” is the “Student Sectioning Problem” (SSP) [3, 2, and others]. In this problem, we are given 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” problems as the SSP, but 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 size, and the constraints, such as those arising from location and timetabling. Our goals are:

- Devise algorithms to do 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

To achieve the above, 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 standard simulated annealing to explore the solution space and deal with the splitting problem.
3. Carry out experiments in order to draw 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 the 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 then 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. 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 most 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 will also apply the condition that the spacetypre of the event must be the same as that of the room. In general, this hard constraint can be softened, and the resulting spacetypre 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 account of models of space utilisation objectives and penalties 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]: The teaching space allocation framework is constrained by timetabling needs, and we believe that space allocation needs to take some account of this. Hence we use here a timetabling penalty (TT) that is just a standard conflict matrix between events; a set of pairs of events that should not be placed at the same timeslot. For this paper we will simply use randomly generated 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 in [13, 2, and others]. For example, 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 [18]). This problem is not studied here, but destined 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 do 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.

Data-set Name:	Wksp	Tut	Sem	Tut-trim
Spacetype	Workshop	Tutorial	Seminar	Tutorial
num. of courses	1077	2088	3711	620
num. of rooms	16	184	88	47
timeslots number	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

Table 1. The four data-sets that we use, and some of their properties, including numbers of rooms and courses, and also the total *Seat-Hours* demanded by all the courses, and the *Seat-Hours* available in all the rooms.

2.3 Overall Objective Function

The overall problem is a multi-objective optimisation problem. However, we work using 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. Using 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 data-set used for [5, 6] as it is not relevant to splitting.)

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 courses to fit without splitting. For **Tut**, the main characteristic of this data-set 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 data-set, **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 data-set, **Sem**, is similar in structure to **Tut**, it exhibits the same characteristics as **Tut**, and

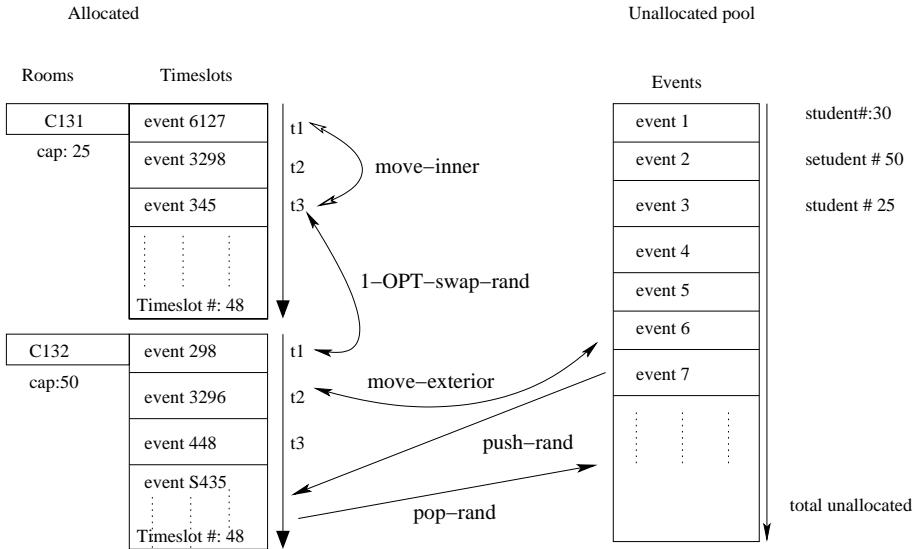


Fig. 1. Schematic of the local search operators (except *2-OPT-swap-rand*) for the local search without splitting.

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-OPT-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-OPT-swap-rand: Similar to *1-OPT-swap* but it randomly selects 4 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.

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 (HC) and Simulated Annealing (SA) [12, 1] 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 new 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. 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 hill-climbing (HC) and the simulated annealing (SA) approaches. Two strategies are implemented: a) constructor-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: Constructor-based static splitting and hill-climbing
- SS-SA: Constructor-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 (hill-climbing or simulated annealing) to improve the initial solution. So, in this strategy, splitting happens within the constructor 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 compare it to the case in which splitting is carried out during the local search (dynamic) which is described in the next subsection.

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 splitting operators, in conjunction with the local search, can discover good solutions that respond not only to the room capacities but also to the penalty values for the location (L), timetabling (TT), section number (SN), section size (SZ), and no partial allocation (NPA). Note that, at the current stage, the operators themselves do not directly respond to penalties, and presumably this leads to inefficiencies because good moves will need to be discovered via multiple attempts within the SA/HC rather than directly and heuristically; 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 elaborateness):

1-OPT-swap-rand-sec: This operator works as **1-OPT-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 as **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 as **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, and 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 δ 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”, δ , 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

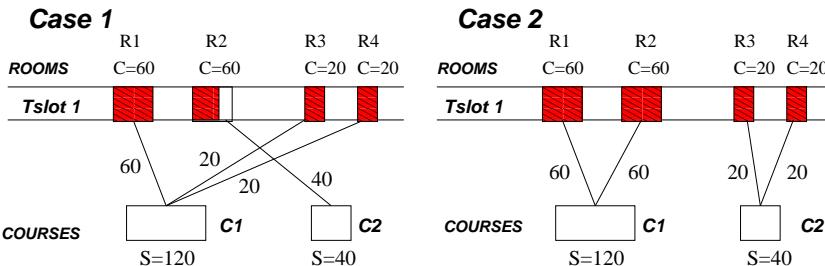


Fig. 2. Example in which applying operators to split courses has different effects. In case 1, course C1 first receives a *push-rand* into room R2, and then applications of *split-push* to C1 are unable to allocate only $60+20+20=100$ students rather than the needed 120. However, in case 2 we see that reversing the order allows all of both courses to be allocated.

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; and 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. But this inevitably means that 20 spaces within room R2 are wasted, and so it becomes impossible to allocate all of course C1. However, 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; though there

are other cases in which ***push-rand*** is necessary. For this reason, and also via experiments, we tend to give the operator ***split-max*** more probability of being selected than the operator ***push-rand***.

4.4 Controlling the search

The example above, and the resulting preference for ***split-max*** over ***push-rand***, is just one case of the standard difficult problem of selecting the operator probabilities.

We have also observed, in an informal manner, that the effectiveness of each operator varies during the search. As an example, suppose we are just doing non-splitting local search from section 3. We start with an empty allocation, and then the ***Push-rand*** operator is 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 constructor does any splitting and is followed by local search, CONS-SA (CONS-HC is not presented as, unsurprisingly, it performs no better than CONS-SA). 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 will see that the DS-SA variant is the better.

However, we first answer the simple question of whether or not, for the data sets that we use, we need to do any splitting at all. 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 constructor) 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 Subsection 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 bit 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.

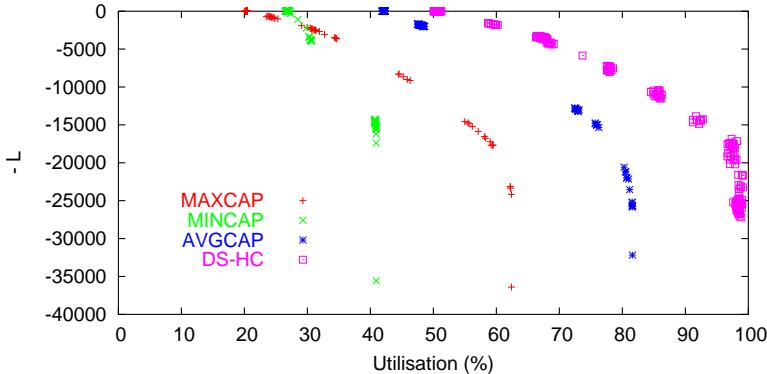


Fig. 3. Comparison of dynamic and static (constructor-based) splitting for the Wksp data set. Plots give the trade-offs obtained between utilisation and Location; all the other objectives being disregarded ($W_{TT}=W_{SZ}=W_{SN}=W_{NPA}=0$). The first three sets of points are from the three constructive methods of subsection 4.1 ; the last “DS-HC” from the dynamic splitting with hill-climbing.

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 solver with those weights. For example, we often plot the trade-off between Utilisation, U , and the location, L ; in this case, we pick a non-zero value for $W(U)$, and then just solve 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 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 subsection 4.1), and compares them to the results from the dynamic splitting method, DS-HC.

We see that for the constructor, 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 constructor-based splitting methods are easily outperformed by the dynamic splitting. This is unsurprising, as it is

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

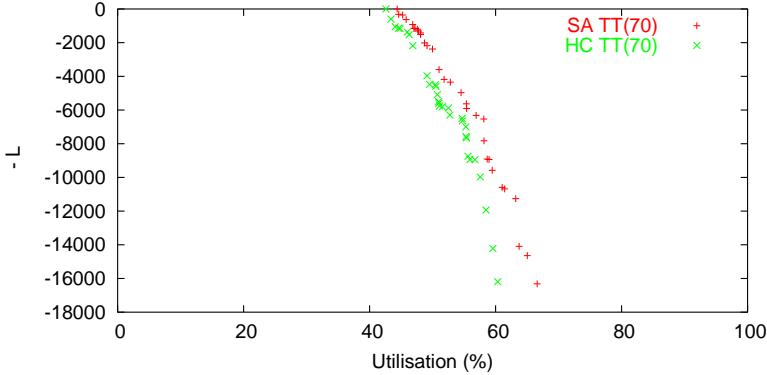


Fig. 4. Trade-off of utilisation and location as obtained with dynamic splitting, and using the hill-climbing (HC) and simulated annealing (SA) algorithms. For the Wksp data, and in the presence of TT(70), and no other constraints beside U, L, and TT.

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 that the performances in the absence of TT are often very similar. Presumably, without the TT, the search space is rather well-behaved.

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 almost entirely on the solution space. In particular, we present some preliminary and partial results on how the various objectives interact, and in particular the magnitude of their effect on the utilisation.

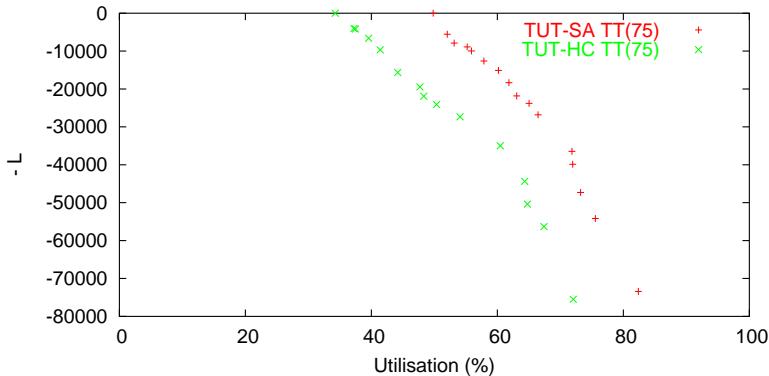


Fig. 5. Same as for Figure 4 but instead using the tutorials dataset, Tut-trim, and with TT(75).

6.1 Interaction of Section Size Penalty (SZ), Location Penalty (L), and Utilisation (U)

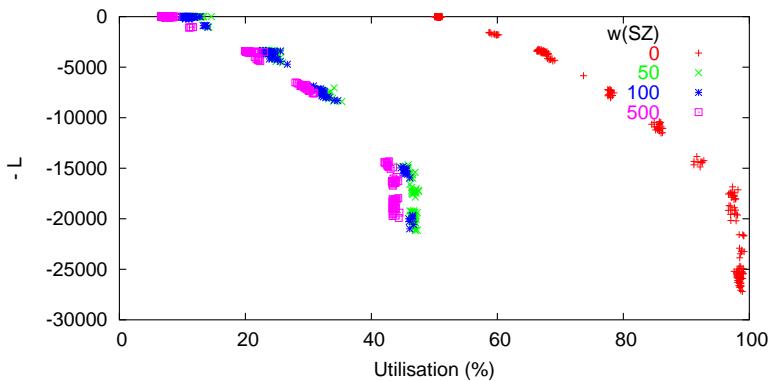


Fig. 6. Trade-off surfaces for the given values of the weight $W(SZ)$ for the section size policy. On the Wksp data-set, with a target section size of 25; and optimising only utilisation U, location L, and section size SZ.

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.

Interestingly, the Pareto front shape seems unaltered by changing the target size, though we currently have no explanation of this, and we believe this issue deserves further investigation.

6.2 Trade-offs Arising From Section Size Penalty and Utilisation

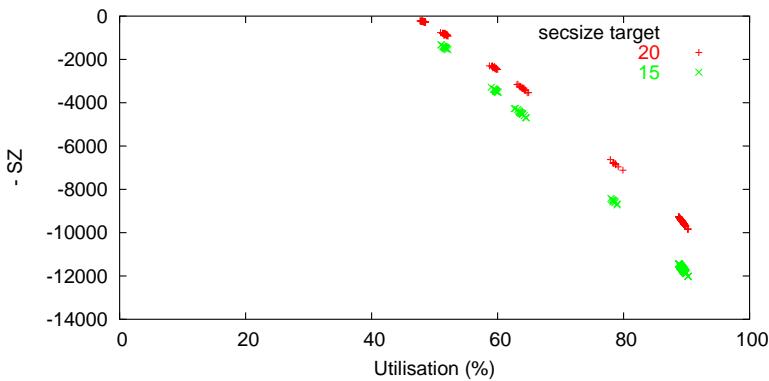


Fig. 7. Utilisation vs. section size penalty, SZ, for the Wksp data set, and for two values(15 and 20) of the target section size.

So far, we have only looked at trade-offs between Utilisation and Location, but 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. In future work, we intend to allow more relaxed and flexible versions of the section size penalty. However, intuitively, it still seems very likely that section size requirements are going to have a strong negative effect on utilisation, and crucially, the methods that we are developing will still allow one to quantify these effects.

Generally, enforcing a soft section size penalty is more realistic than a hard one since flexibility in the section size is quite reasonable; sections aren't always

standardized towards a fixed, unchangeable target size and it ought to be possible to vary the target size to suit other constraints.

6.3 Effects of Timetabling Constraints

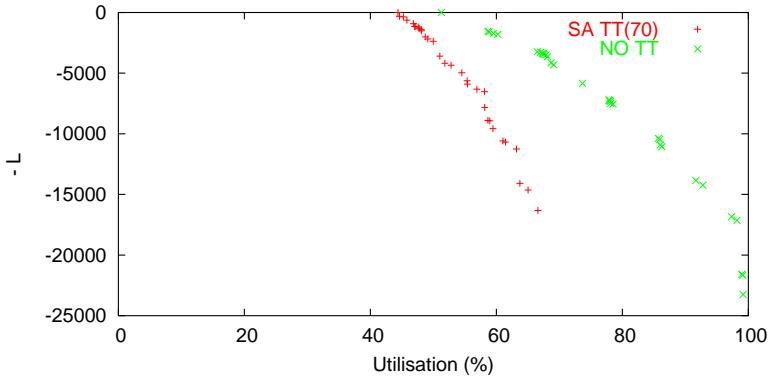


Fig. 8. Trade-offs between Utilisation and Location for the Wksp dataset. “No TT” means that no objectives besides L and U are weighted, in particular $W(TT)=0$. In contrast, “TT(70)” means that a timetabling constraint, with a density of 70% is enforced as a hard constraint.

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 easily 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 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, though 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.

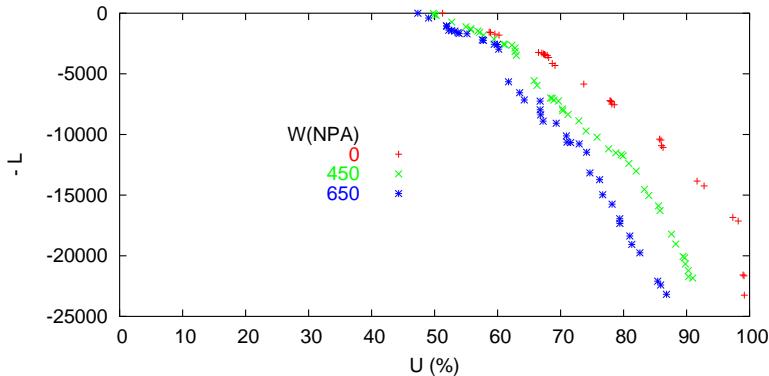


Fig. 9. Trade-offs between Utilisation and Location, in the presence of various strengths of the “No Partial Allocation” (NPA) penalty, but with no TT or other penalties.

7 Summary and Future Work

We have devised methods, and performed preliminary studies of them, to support space planning and space planning in the presence of courses that will need to be split down into multiple sections.

The work broadly splits into two 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, “*dynamic splitting*”, 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.

In future work, we intend to improve the speed and scope of the methods. This will have multiple aspects, but 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. All our inheritance is either total or none. That is, all sections inherit either all conflicts of the associated course, or else they inherit none (equivalent to simply turning off the timetable penalty). Although a defi-

ciency, 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. E. Aarts and J. Korst. *Simulated Annealing and Boltzman Machines*. Wiley, 1990.
2. R. Alvarez-Valdes, E. Crespo, and J.M. Tamarit. Assigning students to course sections using tabu search. *Annals of Operations Research*, 96:1–16, 2000.
3. M. AminToosi, H. Sadoghi Yazdi, and J. Haddadnia. Fuzzy student sectioning. In *Proc. of Practice and Theory of Automated Timetabling (PATAT)*, 2004.
4. V.A. Bardadym. Computer-aided school and university timetabling: The new wave. In *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, volume 1153, pages 22–45. Lecture Notes in Computer Science, Springer-Verlag, 1996.
5. C. Beyrouthy, E.K. Burke, B. McCollum, P. McMullan, J.D. Landa-Silva, and A.J. Parkes. Towards improving the utilisation of university teaching space. Technical Report. School of Computer Science & IT, University of Nottingham, 2006.
6. C. Beyrouthy, E.K. Burke, B. McCollum, P. McMullan, J.D. Landa-Silva, and A.J. Parkes. Understanding the role of UFOs within space exploitation. In *Proceedings of PATAT*, 2006.
7. E.K. Burke, K.S. Jackson, J.H. Kingston, and R.F. Weare. Automated timetabling: The state of the art. *The Computer Journal*, 40(9):565–571, 1997.
8. E.K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002.
9. M.W. Carter. Timetabling. In *Encyclopedia of Operations Research and Management Science*, pages 833–836. Kluwer, 2001.
10. M.W. Carter and G. Laporte. Recent developments in practical course timetabling. In *Selected papers from the second International Conference on Practice and Theory of Automated Timetabling*, pages 3–19. Springer-Verlag, 1998.
11. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19:151–162, 1985.
12. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
13. G. Laporte and S. Desroches. The problem of assigning students to course sections in a large engineering school. *Comput. Oper. Res.*, 13(4):387–394, 1986.
14. B. McCollum and P. McMullan. The cornerstone of effective management and planning of space. Technical report, Realtime Solutions Ltd, Jan 2004.
15. B. McCollum and T. Roche. Scenarios for allocation of space. Technical report, Realtime Solutions Ltd, 2004.
16. S. Petrovic and E.K. Burke. University timetabling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 45. Chapman Hall/CRC Press, 2004.
17. A. Schaefer. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):18–27, 1999.
18. S.M. Selim. Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem. *The Computer Journal*, 31(1):76–82, 1988.

An Experimental Study on Hyper-heuristics and Exam Timetabling

Burak Bilgin, Ender Özcan, 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 nonproblem-specific data, such as, fitness change or heuristic execution time. A typical iteration of a hyper-heuristic algorithm consists of two phases: 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 fourteen well-known benchmark functions and twenty-one exam timetabling problem instances.

1 Introduction

The term hyper-heuristic refers to a recent approach used as a search methodology [2, 3, 5, 11, 21]. It is a higher level of abstraction than metaheuristic methods. Hyper-heuristics involve an iterative strategy that chooses a heuristic to apply to a candidate solution of the problem at hand, at each step. Cowling et al. discusses properties of hyper-heuristics in [11]. 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, 21]. However, no comprehensive study exists that compare the performances of these different mechanisms in depth.

Timetabling problems are real world constraint optimization problems. Due to their NP complete nature [16], traditional approaches might fail to generate a solution to a timetabling problem instance. 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 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, thirty-five hyper-heuristics

generated by coupling all heuristic selection methods and all acceptance criteria with each other, are evaluated on a set of twenty-one exam timetabling benchmark problem instances, including Carter's benchmark [10] and Ozcan's benchmark [25].

The remainder of this paper is organized as follows. In Section 2 background is provided including hyper-heuristics, benchmark functions and exam timetabling. Experimental settings and results for benchmarks are given in Section 3. Hyper-heuristic experiments on exam timetabling are presented in Section 4. Finally, conclusions are discussed in Section 5.

2 Preliminaries

2.1 Hyper-heuristics

Hyper-heuristic methods are described by Cowling et al. [11] as an alternative method to meta-heuristics. 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 general optimization methods, which can be applied to any optimization problem 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 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. The tabu list is emptied every time there is a change in the fitness of the candidate solution [5].

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 Case-Based Heuristic Selection Approach to the 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. First approach, named as *Linear Monte Carlo (LMC)*, uses a negative linear ratio of the probability of acceptance to the fitness worsening. Second approach named as, *Exponential Monte Carlo (EMC)*, uses a negative exponential ratio of the probability of acceptance to the fitness worsening. Third approach, named as *Exponential Monte Carlo with Counter (EMCQ)*, is an improvement over *Exponential Monte Carlo*. 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 simple random mechanism [2].

Kendall and Mohamad [21] 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 heuristic selec-

tion method. In the *Great Deluge Algorithm* initial fitness is set as 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 [21].

Gaw et al. [17] 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 [17].

According to this survey it is concluded that several heuristic selection methods and acceptance criteria are introduced for hyper-heuristics framework. Each pair of the 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, 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, fourteen 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* [18], [19] 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 presented in Tab. 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.

Tab. 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/	lb	ub	opt	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	0	Continuous	Multimodal
Rastrigin, [28]	-5.12	5.12	0	Continuous	Multimodal
Schwefel, [29]	-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, [30]	-	-	0	Discrete	-

2.3 Exam Timetabling

Burke et al. [4, 6] applied a light or a heavy mutation, randomly selecting one, followed by a hill climbing method. Investigation of various combinations of Constraint Satisfaction Strategies with GAs for solving exam timetabling problems can be found in [22]. Paquete et. al. [27] applied a multiobjective evolutionary algorithm (MOEA) based on pareto ranking for solving exam timetabling problem in the Unit of Exact and Human Sciences at University of Algarve. Two objectives were determined as to minimize the number of conflicts within the same *group* and the conflicts among different *groups*. Wong et. al. [32] used a GA utilizing a non-elitist replacement strategy to solve a single exam timetabling problem at 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 Schaefer [14] analyzed tabu search approach using graph coloring based heuristics. Merlot et al. [23] 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. [28] introduced a case based reasoning system to create initial solutions to be used by 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. Ozcan and Ersoy [25] used a

violation directed adaptive hill climber within a memetic algorithm to solve exam timetabling problem. A Java tool named FES is introduced by Ozcan in [26] which utilizes XML as input/output format.

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 exam timetabling problem differs from institution to institution. In this study, three constraints are defined and used as described in [25]:

- (i) A student cannot be scheduled to two exams at the same time slot.
- (ii) If a student is scheduled to two exams in the same day, these should not be assigned to consecutive timeslots.
- (iii) 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 makes 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 [24]. *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 at each heuristic call [24].

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 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' 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. 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 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 t-test. Each combination has been given a ranking. Confidence interval is set to 95% in 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 criteria. 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 depicted in Fig. 1, and the average ranking of each acceptance criterion on all of the functions in Fig. 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 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*. *MC* acceptance criterion performs well on *Foxhole Function*. *IE* acceptance criterion indicates significantly a better performance than the remaining acceptance criteria on average.

Tab. 2. Average ranking of each selection method on each problem; *CF* stands for *Choice Function*, *SR* for *Simple Random*, *RD* for *Random Descent*, *RP* for *Random Permutation*, *RPD* for *Random Permutation Descent*, *TABU* for *Tabu Search*, *GR* for *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 w/ noise	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

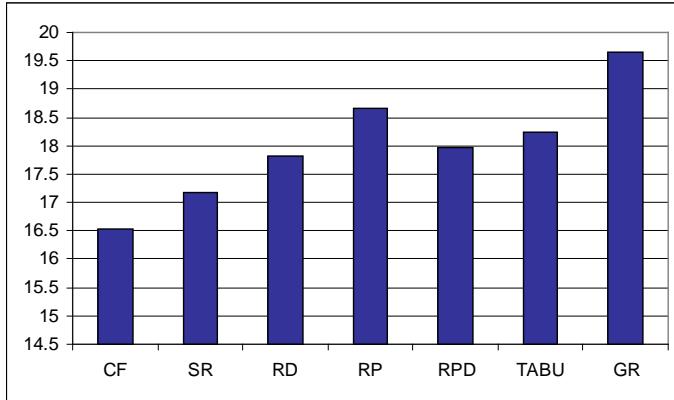


Fig. 1. Average ranking of each selection method on all problem instances

In Fig. 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's Functions*. Fig. 3 (a), (c), and (e) visualize the performance comparison of the heuristic selection methods using *IE* acceptance criterion

for *Sphere*, *Ackley* and *Goldberg's Functions* respectively and Fig. 3 (b), (d), and (f) the performance comparison of the acceptance criteria using Choice Function heuristic selection method for *Sphere*, *Ackley* and *Goldberg's Functions* respectively. Lower average number of evaluations intends faster convergence to the global optimum and indicates better performance.

Table 3. Average ranking of each acceptance criterion on each problem; *AM* stands for *All Moves Accepted*, *OI* for *Only Improving Moves Accepted*, *IE* for *Improving and Equal Moves Accepted*, *MC* for *Monte Carlo Acceptance Criterion*, and *GD* for *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 w/ noise	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

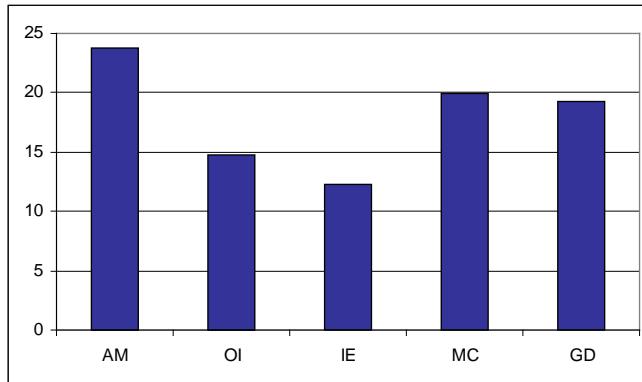


Fig. 2. Average ranking of each acceptance criterion on all problem instances

For *Sphere Model*, distinct performance variances are observed between heuristic selection methods in Fig. 3 (a) on the other side the difference is not so prominent between acceptance criteria in Fig. 3 (b). Fig. 3 (a) shows that Random Permutation and Choice Function heuristic selection methods achieved faster convergence than remaining selection methods. In Fig. 3 (c) and (d) it can be observed that Choice Function heuristic selection method and IE acceptance criterion accomplished a faster

convergence to global optimum on *Ackley Function*. Fig. 3 (e) and (f) show that Choice Function heuristic selection method and IE acceptance criterion performed best on *Goldberg's Function*. Fig. 3 (f) shows that the performance variances between different acceptance criteria are enormous on the same function. Also AM acceptance criterion cannot reach the global optimum on *Goldberg's Function* and no average number of evaluations to converge to global optimum value is depicted for this criterion in the same figure.

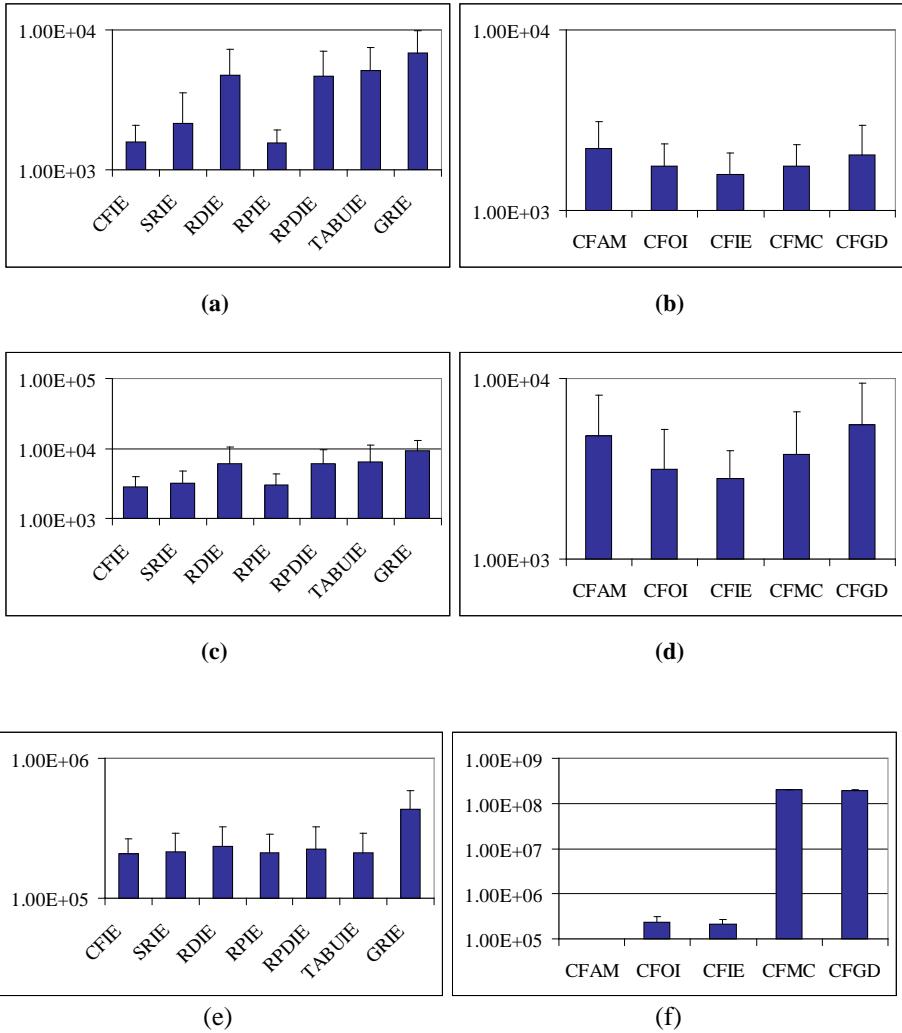


Fig. 3. Average number of evaluations to converge to global optimum of hyper-heuristics consisting of all heuristic selection methods using IE acceptance criterion on (a) *Sphere Model* function, (c) *Ackley Function* (e) *Goldberg Function*, and average number of evaluations to converge to global optimum 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*.

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 hyper-heuristics. The characteristics of as illustrated in Tab. 4.

Tab. 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
Hechs92	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

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.

$$F(T) = \frac{-1}{1 + \sum_{\forall i} w_i g_i(T)} \quad (1)$$

In the 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.

4.1 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 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 conflict 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 conflict 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. Mutational heuristic passes over each exam in the array and assigns a random timeslot to the exam with a predetermined probability ($1/\text{number of courses}$).

4.2 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 to a ranking. Confidence interval is set to 95% in 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.

Average best fitness values for best performing heuristic selection-acceptance criterion combination are provided in Table 5. If several hyper-heuristics share the same ranking, than only one of them appears in the table, marked with *. Seven combinations that have the top average rankings are presented in Fig. 4. According to the results, 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.

Tab. 5. Average best fitness values for best performing heuristic selection-acceptance criterion combinations on each problem instance; *AM* stands for *All Moves Accepted*, *OI* for *Only Improving Moves Accepted*, *IE* for *Improving and Equal Moves Accepted*, *MC* for *Monte Carlo Acceptance Criterion*, *GD* for *Great Deluge Acceptance Criterion*.

Instance	(Av. B. Fit., Std. Dev.)	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
Hechs92	(-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

Tab. 6. The performance rankings of each heuristic selection-acceptance criterion 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

(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
(c)						
H.-h.	Y011	Y012	Y013	Y021	Y022	Y023
SR_AM	26	26	22.5	26	26	9.5
SR_OI	19.5	19.5	31.5	19.5	16	17.5
SR_IE	12	11.5	14	12	12	17.5
SR_MC	6	11.5	4	8	7.5	3.5
SR_GD	1	1	8	1	7.5	7.5
RD_AM	31	31	22.5	03	29.5	9.5
RD_OI	19.5	19.5	31.5	19.5	20	17.5
RD_IE	12	11.5	14	12	12	17.5
RD_MC	6	5	4	4.5	4	1.5
RD_GD	31	31	22.5	30	29.5	9.5
RP_AM	31	31	22.5	34.5	34.5	34.5
RP_OI	19.5	19.5	31.5	19.5	20	28
RP_IE	12	11.5	14	12	12	17.5
RP_MC	6	5	4	4.5	4	25
RP_GD	31	31	22.5	34.5	34.5	34.5
RPD_AM	31	31	22.5	30	29.5	31.5
RPD_OI	19.5	19.5	31.5	19.5	20	28
RPD_IE	12	11.5	14	12	12	17.5
RPD_MC	6	5	4	4.5	4	25
RPD_GD	31	31	22.5	30	29.5	31.5
CF_AM	31	31	22.5	30	33	9.5
CF_OI	19.5	19.5	31.5	19.5	20	17.5
CF_IE	12	11.5	14	12	12	17.5
CF_MC	3	5	4	4.5	1	1.5
CF_GD	19.5	19.5	31.5	19.5	20	17.5
TABU_AM	31	31	22.5	30	29.5	31.5
TABU_OI	19.5	19.5	31.5	19.5	20	28
TABU_IE	12	11.5	14	12	12	17.5
TABU_MC	6	5	4	4.5	4	25
TABU_GD	31	31	22.5	30	29.5	31.5
GR_AM	24.5	24.5	9.5	24.5	24.5	5.5
GR_OI	19.5	19.5	31.5	19.5	20	17.5
GR_IE	12	11.5	14	12	12	17.5
GR_MC	2	2	4	4.5	4	3.5
GR_GD	24.5	24.5	9.5	24.5	24.5	5.5

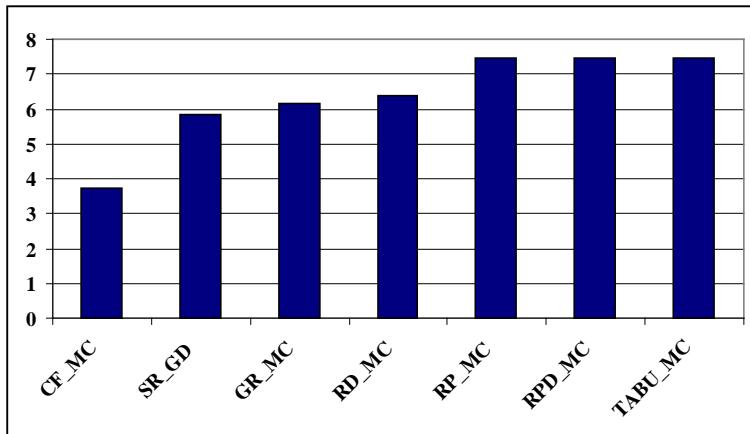


Fig. 4. Top seven heuristic selection method-acceptance criterion combinations considering the average ranking over all problem instances.

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 expanded 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 performance of *Choice Function* and other heuristic selection methods were not as significant as it was between acceptance criteria. The experimental results on exam timetabling benchmark indicated that *Choice Function* heuristic selection method combined with *MC* acceptance criterion performs superior than 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. *Genetic Algorithms and Simulated Annealing*, (1987) 170-215
2. Ayob, M. and Kendall, G.: A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine. *Proceedings of the International Conference on Intelligent Technologies*, InTech'03, Chiang Mai, Thailand, Dec 17-19 (2003) 132-141
3. Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S.: Hyper-heuristics: an Emerging Direction in Modern Search Technology. *Handbook of Metaheuristics* (eds Glover F. and Kochenberger G. A.) (2003) 457-474
4. Burke, E., Newall, J. P., and Weare, R.F.: A Memetic Algorithm for University Exam Timetabling. *Lecture Notes in Computer Science* 1153 (1996) 241-250
5. Burke, E.K., Kendall, G., and Soubeiga, E: A Tabu-Search Hyper-heuristic for Timetabling and Rostering. *Journal of Heuristics* Vol 9, No. 6 (2003) 451-470
6. Burke, E., Elliman, D., Ford, P., and Weare, B.: Examination Timetabling in British Universities- A Survey. *Lecture Notes in Computer Science*, Springer-Verlag, vol. 1153 (1996) 76-90
7. Burke, E.K. and Newall, J.P. : Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings: Models and Algorithms for Planning and Scheduling Problems. *Annals of Operations Research*, vol. 129 (2004) 107-134
8. Burke, E.K., Meisels, A., Petrovic, S. and Qu, R.: A Graph-Based Hyper Heuristic for Timetabling Problems. Accepted for publication in the European Journal of Operational Research (2005)
9. Burke E.K., Petrovic, S. and Qu, R.: Case Based Heuristic Selection for Timetabling Problems. Accepted for publication in the Journal of Scheduling, Vol.9 No2. (2006)
10. Carter, M. W, Laporte, G., and Lee, S.T.: Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the Operational Research Society*, 47 (1996) 373-383
11. Cowling P., Kendall G., and Soubeiga E.: A Hyper-heuristic Approach to Scheduling a Sales Summit. *Proceedings of In LNCS 2079, Practice and Theory of Automated Timetabling III : Third International Conference*, PATAT 2000, Konstanz, Germany, selected papers (eds Burke E.K. and Erben W) (2000) 176-190
12. Davis, L.: Bit Climbing, Representational Bias, and Test Suite Design, Proceeding of the 4th International conference on Genetic Algorithms (1991) 18-23
13. De Jong, K.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan (1975)
14. Di Gaspero, L. and Schaerf, A.: Tabu Search Techniques for Examination Timetabling. *Lecture Notes In Computer Science*, selected papers from the Third International Conference on Practice and Theory of Automated Timetabling (2000) 104 - 117.
15. Easom, E. E.: A Survey of Global Optimization Techniques. M. Eng. thesis, Univ. Louisville, Louisville, KY (1990)
16. Even, S., Itai, A., and Shamir, A.: On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. Comput.*, 5(4):691-703 (1976)
17. Gaw, A., Rattadilok P., and Kwan R. S. K.: Distributed Choice Function Hyperheuristics for Timetabling and Scheduling. *Proc. of the 5th International Conference on the Practice and Theory of Automated Timetabling* (2004) 495-498
18. Goldberg, D. E.: Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction. *Complex Systems* (1989) 129-152
19. Goldberg, D. E.: Genetic Algorithms and Walsh Functions: Part II, Deception and Its Analysis. *Complex Systems* (1989) 153-171

20. Griewangk, A.O.: Generalized Descent of Global Optimization. *Journal of Optimization Theory and Applications*, 34: 11.39 (1981)
21. Kendall G. and Mohamad M.: Channel Assignment in Cellular Communication Using a Great Deluge Hyper-heuristic, in the Proceedings of the 2004 IEEE International Conference on Network (ICON2004)
22. Marin, H. T.: Combinations of GAs and CSP Strategies for Solving Examination Timetabling Problems. Ph. D. Thesis, Instituto Tecnologico y de Estudios Superiores de Monterrey (1998)
23. Merlot, L.T.G., Boland, N., Hughes, B. D., and Stuckey, P.J.: A Hybrid Algorithm for the Examination Timetabling Problem. Proc. of the 4th International Conference on the Practice and Theory of Automated Timetabling (2002) 348-371
24. Mitchell, M., and Forrest, S.: Fitness Landscapes: Royal Road Functions. *Handbook of Evolutionary Computation*, Baeck, T., Fogel, D., Michalewiz, Z., (Ed.), Institute of Physics Publishing and Oxford University (1997)
25. Ozcan, E., and Ersoy, E.: Final Exam Scheduler - FES, Proc. of 2005 IEEE Congress on Evolutionary Computation, vol. 2, (2005) 1356-1363
26. Ozcan, E., Towards an XML based standard for Timetabling Problems: TTML, *Multidisciplinary Scheduling: Theory and Applications*, Springer Verlag, (2005) 163 (24)
27. Paquete, L. F. and Fonseca, C. M.: A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms. Proc. of the 4th Metaheuristics International Conference (MIC 2001) 149-154
28. Petrovic, S., Yang, Y., and Dror, M.: Case-based Initialisation for Examination Timetabling. Proc. of 1st Multidisciplinary Intl. Conf. on Scheduling: Theory and Applications (MISTA 2003), Nottingham, UK, Aug 13-16 (2003) 137-154
29. Rastrigin, L. A.: Extremal Control Systems. In *Theoretical Foundations of Engineering Cybernetics Series*, Moscow, Nauka, Russian (1974)
30. Schwefel, H. P.: Numerical Optimization of Computer Models, John Wiley & Sons (1981), English translation of *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* (1977)
31. Whitley, D.: Fundamental Principles of Deception in Genetic Search. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Matco, CA (1991)
32. Wong, T., Côté, P. and Gely, P.: Final Exam Timetabling: A Practical Approach. Proc. of IEEE Canadian Conference on Electrical and Computer Engineering, Winnipeg, CA, May 12-15, vol. 2 (2002) 726-731

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.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 these informations (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 did not yield acceptable results anymore.

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 present an elegant integer linear programming model for this problem that easily can be put into CPLEX. Finally, we report on our computational experiments and results around the Eindhoven real-world data.

Keywords: University timetabling; network flow formulation; NP-completeness; integer programming formulation.

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 these 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 4 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 settle 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 are 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 will 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 time tabling. 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 will be 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 & Ströhlein [12] lists many papers that appeared before 1980. Schaerf [11] gives a survey of the various formulations of timetabling problems and classifies the timetabling problem into the following three main classes:

School timetabling: The weekly scheduling of all the classes of a high school.

Avoid that teachers meet two classes at the same time, and avoid that classes meet two teachers at the same time.

Examination timetabling: The scheduling of the exams of several university courses. Avoid that exams of courses with common students overlap. Spread out the exams for every student as much as possible over time.

Course timetabling: The weekly scheduling for all the lectures of several university courses. Minimize the overlaps of lectures of courses with common students.

Of course this classification is crude, and there are many real-world timetabling problems that fall in between two of these classes.

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, Itai & Shamir [5] 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, Itai & Shamir [5], Garey & Johnson [7] and de Werra [4].

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. Schaefer [11] gives an integer linear programming formulation of the examination timetabling problem and describes some alternative variants of the problem.

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 [4] gives a binary integer programming formulation. Schaefer [11] discusses some of the most common variants of the basic formulation.

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 [2], Feldman & Golumbic [6] and Laporte & Desrochers [8].

Cheng, Kruk & Lipman [3] discuss the Student Scheduling Problem (SSP) as it generally applies to high schools in North America. They define the problem as the assignation of courses and a specific section to each student. The objective is to fulfil 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 in most practical cases into an empty feasible solution set.

Laporte & Desrochers [8] 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 [13] formulated the student scheduling problem as an integer linear programming problem and uses Lagrangian Relaxation to solve it. Sabin & Winter [10] use a greedy approach that is moderated by an intelligent ordering of the students. Miyaji, Ohno & Mine [9] apply goal programming.

3 Problem Description

At our first meeting, the Industrial Design department explained the problem to us in a certain problem formulation No. 1; see Subsection 3.1. This problem can be modeled as a network flow problem, and hence is solvable in polynomial time; see Ahuja, Magnanti & Orlin [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. Subsection 3.2 describes formulation No. 2.

3.1 Problem Formulation No. 1

At the first meeting with the Industrial Design department, they told us 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 can not do two courses at the same time and there is a maximum number of students that can be assigned to a course. This subsection 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. For each course also the weekly meeting time is already assigned. 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 can not 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 and 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 .

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

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 its 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

As 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 can not 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. The remainder of this subsection explains these extra restrictions and defines the problem into more detail.

An academic year is divided into a certain number of periods. The length of such a period depends on the number of periods in which the academic year is split. For instance, the academic year 2005-2006 is divided into six periods of five weeks. We define such a period as a *block*. The Industrial Design department wants us to schedule two 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. This can not be modeled with a flow constraint. 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 already 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. 1, r_s was defined as the requested number of courses of student s . This definition is adjusted in problem formulation No. 2 into the requested workload of student s for $|B|$ blocks together. For every student s , a maximum requested workload r_{sb} for each block $b \in B$ separately is given, because the requested workload of a student is not always equally divided over all blocks $b \in B$. For instance, if blocks b_1 and b_2 have to be scheduled simultaneously and $r_s = 3$, then parameters r_{sb_1} and r_{sb_2} are both equal to 2. In this case the model is allowed to choose the block in which student s is assigned two courses. Another example, 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, hence have two time slots. If such a course is assigned to a student, the student has to be available at both time slots. If courses with two time slots are introduced into problem formulation No. 1, the problem can not be modeled as a network flow problem.

The set C of courses offered to the students contains courses with multiple sections, meaning that the course is repeated during the week. Table 2 contains course DA242 as an example. The time slots in the table are encoded. For example, code B1TM2 stands for the second part of Tuesday morning in block 1. The workloads of a course in block 1 and 2 are denoted with $wlb1$ and $wlb2$. The course DA242 has five sections which all have two time slots as meeting times. The first meeting is for all sections on the same time slot and the other is on a different time slot for each section. The first meeting is a class in one large lecture room and the second is a meeting where exercises have to be made in smaller groups.

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. In problem formulation No. 2 there are

no maximum and minimum number of students for a course like in problem formulation No. 1, but a minimum number C_i^{min} and a maximum number C_i^{max} of students for each section $i \in I$. 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 $T(i) = \emptyset$.

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

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 .

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 needs to be assigned to at least a certain minimum number of students and student s has to be assigned to course c_{sp} if $(s, p) \in U$.

4 The Integer Linear Programming Model

To build a schedule which best fits the needs for the students, the problem is split into four subproblems which are formulated as an integer linear programming problem. These subproblems are solved sequentially, keeping the objective value of the foregoing subproblems the same. 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 don't 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:

$$x_{sp} = \sum_{i \in I | c_{sp} = c(i)} z_{spi} \quad \forall s \in S, \forall p \in P_s \quad (1)$$

$$\sum_{p \in P_s} \sum_{i \in I | c_{sp} = c(i)} w(c_{sp}, b) z_{spi} \leq r_{sb} \quad \forall s \in S, \forall b \in B \quad (2)$$

$$\sum_{p \in P_s} \sum_{i \in I | c_{sp} = c(i)} \sum_{b \in B} w(c_{sp}, b) z_{spi} \leq r_s \quad \forall s \in S \quad (3)$$

$$\sum_{s \in S} \sum_{p \in P_s, c_{sp} = c(i)} z_{spi} \leq C_i^{\max} y_i \quad \forall i \in I \quad (4)$$

$$\sum_{p \in P_s} \sum_{i \in I | c_{sp} = c(i), t \in T(i)} z_{spi} \leq 1 \quad \forall s \in S, \forall t \in T \quad (5)$$

$$\begin{aligned} z_{spi} &= 1 & \forall s \in S, \forall p \in P_s, \\ & & \forall i \in I | (s, p, i) \in F \end{aligned} \quad (6)$$

$$x_{sp} \in \{0, 1\} \quad \forall s \in S, \forall p \in P_s \quad (7)$$

$$y_i \in \{0, 1\} \quad \forall i \in I \quad (8)$$

$$z_{spi} \in \{0, 1\} \quad \forall s \in S, \forall p \in P_s, \forall i \in I \quad (9)$$

Constraint (1) takes care 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 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) takes care 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).

As explained above, the problem is split into four subproblems which are solved sequentially. 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) 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 , i.e. the minimum number C_i^{\min} of students subtracted with the number of students assigned to section i . The second subproblem minimizes the total shortage S^{\min} of students. This results into the following ILP formulation:

$$S^{\min} = \min \sum_{i \in I} s_i$$

$$\begin{aligned}
\sum_{(s,p) \in U} x_{sp} &= U^{max} \\
\sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} + s_i &\geq C_i^{min} y_i \quad \forall i \in I \\
s_i &\in \mathbb{Z}^+, \quad \forall i \in I \\
(\text{x,y,z}) &\text{ satisfy (1)-(9)}
\end{aligned}$$

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:

$$\begin{aligned}
W^{max} = \max \sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b) x_{sp} \\
\sum_{i \in I} s_i &= S^{min} \\
\sum_{(s,p) \in U} x_{sp} &= U^{max} \\
\sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} + s_i &\geq C_i^{min} y_i, \quad \forall i \in I \\
s_i &\in \mathbb{Z}^+, \quad \forall i \in I \\
(\text{x,y,z}) &\text{ satisfy (1)-(9)}
\end{aligned}$$

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 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 that one student gets a lot of courses of his 7th up to 10th position of his preference list. A constraint is added to the model that enforces that every student gets at most one course from these positions, else 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 on the 7th up to 10th position of his preference list.

This results into the final ILP formulation:

$$\begin{aligned}
 \min W_p \sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b) (82 - (10 - p)^2) x_{sp} + W_s \sum_{c \in C} S_c + W_e \sum_{s \in S} E_s \\
 I_{c(i)}^{max} \geq \sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} \quad \forall i \in I \\
 S_c = \sum_{i \in I | c=c(i)} (I_c^{max} - \sum_{s \in S} \sum_{p \in P_s, c_{sp}=c} z_{spi}) \quad \forall c \in C \\
 \sum_{p=7}^{10} x_{sp} \leq 1 + E_s \quad \forall s \in S \\
 \sum_{s \in S} \sum_{p \in P_s} \sum_{b \in B} w(c_{sp}, b) x_{sp} = W^{max} \\
 \sum_{i \in I} s_i = S^{min} \\
 \sum_{(s,p) \in U} x_{sp} = U^{max} \\
 \sum_{s \in S} \sum_{p \in P_s, c_{sp}=c(i)} z_{spi} + s_i \geq C_i^{min} y_i, \quad \forall i \in I \\
 E_s \in \mathbb{N} \quad \forall s \in S \\
 I_c^{max}, S_c \in \mathbb{N} \quad \forall c \in C \\
 s_i \in \mathbb{Z}^+ \quad \forall i \in I \\
 (\text{x,y,z}) \text{ satisfy (1)-(9)}
 \end{aligned}$$

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 had to be assigned. 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.

Table 3. Input information for academic year 2005-2006

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

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 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 2005-2006 are given in Table 4. What can be noted is that the computation time of CPLEX is negligible.

Table 4. Results for the academic year 2005-2006

	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

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 10 courses.
- a preference list with not enough different time slots in one of the two blocks.
- a preference list with the same course on more positions. There was even a student with ten times the same course on his preference list.

If all students would hand in a preference list with 10 courses and enough different time slots, then in blocks 1 & 2 only five students would not be assigned to their requested number of courses, 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 6 suitable urgent courses of which at most 4 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. Hence, it can be concluded that students get a lot of courses which are on top of their preference list. A bad position is a course

assigned to a student who has the course on 7th up to 10th position on its preference list. Also from the number of bad positions it can be concluded that the courses assigned to students are on top of their preference lists.

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.

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. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms and applications*. Englewood Cliffs: Prentice Hall, New Jersey, 1993.
2. V.A. Busam. *An algorithm for class scheduling with section preference*. Communications of the ACM, 10(9), 567–569, 1967.
3. E. Cheng, S. Kruk, and M. Lipman. *Flow formulations for the student scheduling problem*. In Practice and Theory of Automated Timetabling IV, Burke and De Causmaecker, Lecture Notes in Computer Science, Vol. 2740 Springer-Verlag, 2002.
4. D. de Werra. *An introduction to timetabling*. European Journal of Operations Research 19, 151–162, 1985.
5. S. Even, A. Itai, and A. Shamir. *On the complexity of timetable and multicommodity flow problems*. SIAM Journal of Computing 5, 691–703, 1976.
6. R. Feldman and M.C. Golumbic. *Constraint satisfiability algorithms for interactive student scheduling*. In Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 89), 1010–1016, 1989.
7. M.R. Garey and D.S. Johnson. *Computers and intractability - a guide to NP-completeness*. W.H. Freeman and Company, San Fransisco, 1979.
8. G. Laporte and S. Desrochers. *The problem of assigning students to course sections in a large engineering school*. Computational Operations Research 13, 387–394, 1986.
9. I. Miyaji, K. Ohno, and H. Mine. *Solution method for partitioning students into groups*. European Journal of Operations Research 33, 82–90, 1981.
10. G.C.W. Sabin and G.K. Winter. *The impact of automated timetabling on universities - a case study*. Journal of Operations Research Society 37, 689–693, 1986.
11. A. Schaerf. *A survey of automated timetabling*. Artificial Intelligence Review 13(2), 87–127, 1999.
12. G. Schmidt and T. Ströhlein. *Timetable construction - an annotated bibliography*. The Computer Journal 23, 307–316, 1980.
13. A. Tripathy. *Computerised decision aid for timetabling - a case analysis*. Discrete Applied Mathematics 35(3), 313–323, 1992.

A Max-Flow Model of Problem Formulation No. 1

Full details of the definition of this network flow problem will be given in the full version of this paper.

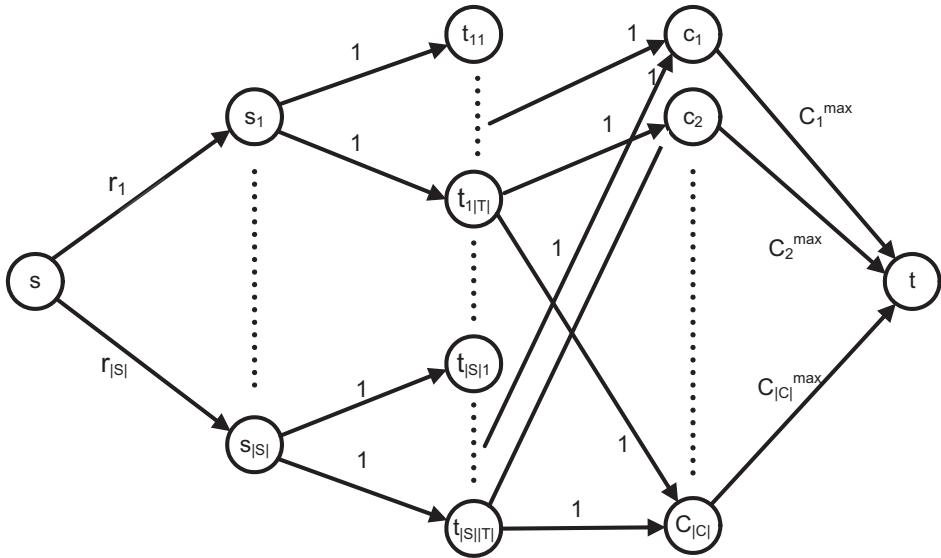


Fig. 1. The network flow model

B Some NP-hardness Results

The timetabling problem defined in Subsection 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 $st(\bar{x}_i)$ 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 $st(\bar{x}_i)$.
- 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 $st(\bar{x}_i)$) if and only if x_i (respectively \bar{x}_i) occurs as a literal in clause c_j .

Note that in any feasible assignment, student $st(x_i)$ (respectively student $st(\bar{x}_i)$) will either do course $C(x_i)$ or the two courses $C(c_{j_1})$ and $C(c_{j_2})$ for which literal x_i (respectively literal \bar{x}_i) occurs in clauses c_{j_1} and c_{j_2} .

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 $st(\bar{x}_i)$ to the two clause-courses that correspond to the clauses containing \bar{x}_i . If x_i is set to FALSE, we assign $st(x_i)$ to the courses that correspond to the clauses containing x_i , and student $st(\bar{x}_i)$ 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-courses 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 $C(c_{j_1})$ and $C(c_{j_2})$. 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

Strategic Employee Scheduling

Peter Chan Y.C.¹, Michael Hiroux^{1,2}, Georges Weil^{1,2}

¹ Equitime S.A., 19 Rue du Granier, 38240 Meylan, France
`{pchan, mhiroux, gweil}@equitime.fr`
<http://www.equitime.fr>

² Laboratory TIMC-CNRS-IMAG, University Joseph Fourier,
38705 La Tronche, France
<http://www-timc.imag.fr>

Abstract. Today's highly competitive economy calls for new methods of management. Advanced practices have been proposed to manage human resources, often acclaimed to be the most important assets of any organisation. However, computer models and applications to support these methods are often not available, or not until it is much too late. This paper presents several directions for advances in strategic employee scheduling, as well as our approach for implementing these concepts.

Introduction

It is commonly observed that human resource (HR) models and applications take time to keep up with emerging management best practices (MBP). For example, the handling of homogeneous employees first published in 1950's, is still subject to research today (see [10, 14]), while scheduling employees with multiple skills have been discussed since 1980's. To accelerate this process, this paper describes in § 1, a consistent set of MBP that would make up a Strategic Employee Scheduling (SES) system. A definition of SES is given in § 2 and we compare it with existing terminology and models. We will describe our approach in implementing such a system in § 3 with some details based on Mixed Integer Programming.

In this paper, we do not consider simpler working conditions where only days-off needs to be scheduled or where the requirements are cyclic, i.e. they repeat systematically after a given period of time, typically weekly. In addition, we do not consider shift creation, which assumes cyclic requirements over a given day of the week.

1 Management Best Practices

This part describes the many concepts that managers need to consider in producing "strategic" schedules, i.e. scheduling with a strategy. Like multi-skilled staff, these concepts are not new. However, computing models and applications that handle them are only partially available today, e.g. [12].

1.1 Creating, Operating and Retaining Flexible Teams

It is common knowledge that multiple skilled workers are more productive since they can change jobs to meet changing customer needs. The underlying principle is *flexibility*; i.e. teams that can *easily* and *quickly* adapt to changing market conditions; see [6]. Through our experience, *operating* a flexible team involves concepts such as:

- **Multi-term:** Annualized hours allow people to work more on certain weeks without incurring overtime. In order to avoid abuse, this flexibility is accompanied by maximal work limits at various horizons (e.g. daily, weekly, monthly or quarterly) and minimal rest duration at the day and week levels. Capacity planning becomes a necessity to avoid paying fines when these limits are violated. This important concept, similar to that of “Planning and Scheduling”, is discussed further in § 1.2. Since 2000, annualized work time has become legal for many sectors of the economy in many European countries. We have been working in this area [4], [5].
- **Multi-contracts:** People may come from different walks of life with different work durations and times, e.g. students, house-wives, retired or semi-retired people. For economic reasons, different populations may be hired to cater for peak periods; their differences may be used to adapt team availability to different customer demands in the day, over the week or over a season (e.g. summer/winter season).
- **Multi-site/multi-project:** People may work on different sites or projects, according to the needs of the moment. Rather than hiring and training new personnel, it might be more efficient to have them travel across sites, e.g. during meal breaks.

In addition to *operation*, the team needs to be *created* and its members *retained*. Team creation involves the identification of key roles within the team, the assignment of available individuals to these roles and the recruitment of new staff for the missing roles. This aspect is out of the scope of the paper, being a one-time activity for which automation may not be cost-effective.

We think that *retaining* team members is an aspect that accompanies team *operation*. Other than better pay, motivation can come by work times adapted to individual needs which can change over time, better working conditions (such as security and hygiene), creation of a team spirit or through *professional mobility*. For example, highly skilled staff can act as tutors to new employees.

1.2 Capacity Planning and Scheduling and Strategic Employee Scheduling

Recently, the concept of Integrated Planning and Scheduling has been introduced, initially in the domain of autonomous systems where actions must be planned using AI-based methods and then scheduled for execution; e.g. see [13]. The constituent domains have been studied separately until recently. Both are highly combinatorial problems and their resolution methods are not dissimilar. Their integration and/or simultaneous execution within the same application are motivated by creating near-perfect schedules, to solve the problems faster, or to solve even larger ones.

We argue that AI-based planning is not always relevant in general management practices. Here, the goals are well defined in advance and do not necessarily evolve over time. In many classical scheduling areas, the concept of planning is typically

based on capacity reasoning. So that given activities can take place as scheduled, it is necessary that all required resources and constituents be present in adequate number, in space and in time.

For example, in manufacturing where machines need to be scheduled (at the job-shop level), Materials Requirements Planning (MRP) software is used to organize activities so that constituent parts are available on time (at the same factory/site) and in required quantities. Resource capacity constraints are normally taken into account in Manufacturing Resource Planning (MRP2) software. Currently, such software's run independently in different departments of the company. Plans that respect capacity can be created in MRP2 that cannot be scheduled in MRP.

We see capacity planning as a natural extension to detailed scheduling, with the goal of ensuring that needed resources and materials be available in time and in quantity so that the schedule can actually be implemented. We expect the capacity planning and scheduling processes would be running step-in-step. Many of the scheduling definitions would come from capacity planning, such as team size, skill compositions, etc. When capacity planning shows that there is excessive unused capacity, it may be empowered to launch new activities.

Within the more general context of flexible teams such as that described in §1.1, we would refer to the Capacity Planning and Scheduling concept as *Strategic Employee Scheduling*, so as to avoid confusion with AI-based Planning and Scheduling.

1.3 Benefit = Revenue - Costs

We see that flexible teams seek to adapt team availability to customer requirements so that requirements can be fulfilled so that the business opportunities are not lost. The underlying concept is that the employees be fully and usefully occupied. In other words, avoid downtime. To avoid downtime, managers launch additional activities or projects. For the same fixed costs, increasing revenue would produce more benefits.

- Detect if there is enough slack to launch a new activity, while allowing for some slack to cater for unforeseen circumstances.
- Choice of a new job/mission/production to introduce/launch; this may depend on availability thresholds
- Which item to make to stock: it depends on available manpower (or what's missing and must be completed by hires), stocks of spare parts and stocking newly assembled parts.
- Compact schedules are those that have work periods (hours/days/weeks) over the shortest possible horizon. Create compact schedules at the highest possible level (e.g. quarterly), so that people can be reassigned elsewhere or on other projects.

2 Strategic Employee Scheduling: A Definition

Strategic Employee Scheduling is the process of producing detailed daily schedules for individual employees while taking the organisation's strategic goals into considerations at different time horizons (such as monthly, quarterly or yearly). This defini-

tion stems from the term “Strategic Scheduling” for example in manufacturing, where lead times for business decisions range from 3 months to several years. Strategic Scheduling is a general management methodology to consider an organization’s strategic goals and scheduling all resources to meet them. With a larger scope, it can achieve more important gains than ordinary scheduling. Models and applications have been proposed but they are *necessarily* specific to the domain or the combination of resources managed (manufacturing, transport, farm production, etc.).

In certain cases, it may be possible to combine two existing models or tools to cover the strategic terms (long and middle) and the short term. Here, a good match is essential because we need both *good strategies that can be scheduled* and *good short-term schedules that are long-sighted*. And we need to get them without having to adjust by hand the results of one to feed the other. In the following paragraphs, we offer a more precise definition in terms of objects and concepts manipulated.

2.1 Scheduling workforces, nurses or employees

Workforce scheduling is taken to be short-term assignment of tasks in time, with the attendant sequencing/precedence constraints. The people scheduled are assumed homogeneous such that individual skills are not taken into account, such as in technologically mature industries. The first work started by [7], scheduling homogeneous workers is still a research subject today [10, 14].

Employee scheduling, a term first used in [9], takes into account individual skills. Distinguishing full-time and part-time employees, each employee specifies the minimum and maximum hours per week and duration and times during the week. In the literature, some authors misuse *workforce* scheduling to refer to *employee* scheduling.

Nurse scheduling is generally more complex, producing subtly “balanced” schedules for each employee according to their individual preferences; see e.g. [1].

Some properties to be taken into account:

- Set of skills and the level of proficiency for each employee. This allows him/her to be assigned to simple tasks in a new skill, thus allowing a gradual development.
- We need to know the employees’ previous assignments so as to ensure minimum rest duration since yesterday’s work or enough rest days in the week, and to ensure that maximum work duration is not exceeded in the current month or quarter. Scheduling history can also be used to produce schedules that are balanced with respect to values of counters (such as number of night and/or weekend assignments).
- Contractual and preferred work periods and durations
- Skill and proficiency level required for each activity type.
- Per skill, the minimum assignment for each employee, thus taking into account his previous assignments. The minimum is to retain the skill qualification (for security reasons) or to upgrade it, depending on the organisation’s policy.
- Company skills to develop, employees designated for training in these skills
- Identification of activities that may be launched and the thresholds of excess manpower that justifies their launching

2.2 Planning model

The capacity planning model is an aggregated model using periods of one day, week, or month, over an annual horizon, for example. The needed amount of work (i.e. man-hours) per skill per period is forecast, either based on statistics with local corrections for events in the new horizon, or activities validated by higher management. Other inputs are employees' absence requests (i.e. summer/winter holidays). At the beginning of the year, such forecasts may be incomplete; requests formulated during the year may not be granted, especially during peak seasons.

Capacity planning consists of determining the work duration per employee per period and per skill. Other results are:

- Planning off-peak seasons where employees can take annual vacations; these assignments are nominative but employees with the same skills may exchange them
- Hiring additional hands when needs cannot be satisfied with available employees
- Launching additional activities when available manpower exceeds requirements by a given margin.

This component gives SES its strategic dimension over pure scheduling systems.

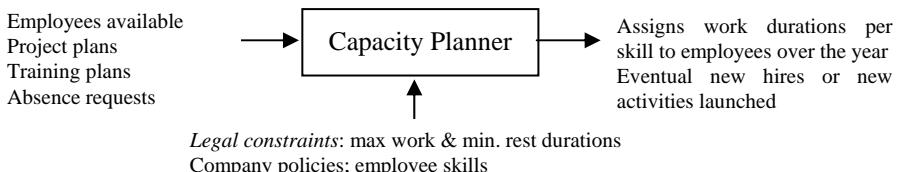


Fig. 1: Capacity Planning

2.3 Scheduling model

The scheduling model is the detailed assignment of employees to activities or skills on each day of the week. The schedule must respect the different daily/weekly constraints on work and rest duration, total work duration and total work duration per skill. There are eventually $\frac{1}{4}$ hourly requirements per day of the week, similar to those in call centers. In the distribution sector, depending on the holidays in the week, a given weekly load curve can be broken down into *standard* load curves per day.

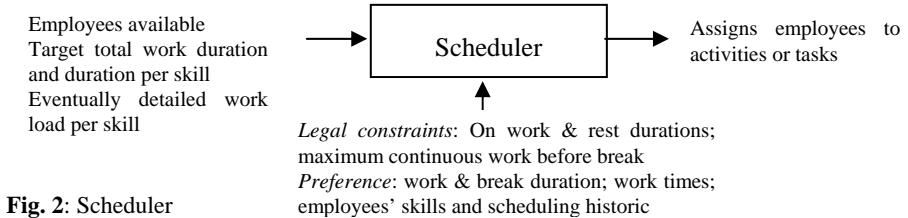


Fig. 2: Scheduler

The scheduler could also be used to verify if various parts of the annual capacity plan can be scheduled. Compared to conventional schedulers, it handles multi-skills and performance levels. It would also take into consideration some planning con-

straints such as slack thresholds: if exceeded, it would abort and request the planner to activate additional tasks (which invalidates the current schedule anyway).

It is the presence of the planning model and its integration to the scheduling model that transforms the whole into an SES. The integration may be at the model level, where the linear equations of both levels coexist.

3 Our Approach to Strategic Employee Scheduling

In this section, we describe our approach to solving the SES problem as described in § 2. We propose two models for (a) capacity planning and (b) detailed scheduling. We solve a capacity planning problem at the annual horizon with weekly periods. Here the work hours of each employee are distributed so as to meet the forecasted demand, i.e. the number of hours of expected work $NW(s, w)$ per skill s and per week w . The total working duration per week is bounded legally. The average working duration per week over 3 months and the total annual working duration are also bounded. Employees' requests for summer / winter holidays may be integrated within the plan at this stage. With the weekly skill distribution known, we attempt to produce a detailed schedule for all days in the current weeks that is compatible with labor constraints such as maximum work and minimum rest durations per day and per week. If such a schedule is unfeasible, we recalculate the annual plan; in particular, we check the plan for the following weeks and eventually recalculate the detailed schedule for some weeks (if they are already calculated).

In the following, we first detail the capacity planning step. The detailed scheduling step can be formulated as described in § 3.2. The set of employees is denoted **Employees**; the skills of employee e is denoted **Skills**(e); the periods in day d (or week w) is noted **Periods**(d) or **Periods**(w). We assume that each skill implies the site at which the skill can be exercised.

The proficiency level of a skill is factored out of the mathematical model. Each combination of (Skill, Site, and proficiency level) is mapped into a different skill, e.g. $s1 = (s^o, \text{Paris}, \text{high})$, $s2 = (s^o, \text{Paris}, \text{medium})$, $s3 = (s^o, \text{Paris}, \text{low})$. An employee expert in s^o will have the three mapped skills and a trainee will have only $s3$.

3.1 Capacity planning in SES

The capacity planning problem uses the integer variables $Y(e, s, w)$ representing the number of hours worked by employee e in a skill s over week w . The total work duration in the week w is given by (1). WD is a semi-continuous variable, bounded by the minimum and maximum contractual weekly work duration: $CW_{\min}(e)$ and $CW_{\max}(e)$. It is null if employee e takes weekly holidays.

$$\begin{aligned} WD(e, w) &= \sum_{s \in \text{Skills}(e)} Y(e, s, w), \forall e, \forall w. \\ CW_{\min}(e) &\leq WD(e, w) \leq CW_{\max}(e). \end{aligned} \tag{1}$$

The legal annual work limit $CA_{\max}(e)$ is assured by

$$\sum_{w=1 \dots 52} WD(e, w) \leq CA_{Max}(e), \forall e. \quad (2)$$

Another legal limit is that the average weekly hours over all sliding horizons of CH=16 consecutive weeks must not exceed CH_{Max}:

$$\sum_{w=0 \dots CH-1} W(e, a+w) \leq CH_{Max}/CH, \forall e, \forall a = 1 \dots 52 - CH \quad (3)$$

The requirements constraint is given by NW(s, w) for skill s in week w:

$$\sum_{e \in Employees} Y(e, s, w) \geq NW(s, w), \forall s, \forall w. \quad (4)$$

Creation of compact plans at the week level

The logical conditions that imply compact schedules (§ 1.3) is that when there is no work on weeks w and w+2, week w+1 must be off as well:

If WD(e, w) > 0 and WD(e, w+2) > 0 **Then** WD(e, w+1) > 0

This condition may be translated into linear equations. We use binary variables Bi to hold when the WD variables > 0; M designates the bound CW_{Max}(e)

If WD(e, w) > 0 **Then** X0 = 1 WD(e, w) ≤ M × B0; M × B0 – M ≤ WD(e, w);

If WD(e, w+2) > 0 **Then** X2 = 1 WD(e, w+2) ≤ M × B2; M × B2 – M ≤ WD(e, w+2);

Next, we take the product of the variables B0 and B2 and link them to WD(e, w+1):

$$B1 \leq B2; B1 \leq B0; B1 - 1 + B0 \leq B1; B1 \leq WD(e, w+1)$$

Hence, over 52 weeks, we add 400 equations and 52 Boolean variables per employee. When single off weeks are requested by the employee, we have to avoid posting the corresponding equations.

New hires

Some dummy employees would be included during capacity planning. The lower bounds on total work duration per week would lead to employees either partially or completely unemployed over the year, which means that they can be removed. If the work load exceeds available work capacity of dummy and real employees, the linear relaxation of the system of equations would quickly prove to be infeasible.

Launching new activities

Projects that last more than a week, with different skill requirements during each week of the project life, need to be scheduled, i.e. assign them in time subject to resource capacity limits. We expect such projects to be decided and scheduled very early in the process and taken in account by NW(s, w). SES needs only to consider launching projects or activities that can be completed within the week, given enough manpower.

To do so, capacity planning is activated without extra employees. At week w, the selection of projects to launch is a classical project selection problem with the 0-1 variables Project(j, w) = 1 if project j is selected for the week w, 0 otherwise. Given a set of **Projects**, where each j requires a(j, s) hours of skill s, the basic requirement is

$$\sum_{j \in Projects} a(j, s) \times Project(j, w) \leq NW(s, w) - \sum_{e \in Employees} Y(e, s, w), \forall s, w$$

The objective function to maximize is $\sum_{j \in \text{Projects}} c(j) \times \text{Project}(j, w)$, $c(j)$ being the profit of the project j . These projects are selected and added to $\text{NW}(s, w)$ before moving onto the scheduling step.

Hence, handling new activities is not a direct MIP problem but requires updating the weekly requirements, may require user interaction to finalize the selected projects.

3.2 Scheduling with 0-1 variables and patterns

The scheduling problem of day d uses the 0-1 variables: $X(e, s, p)$ takes the value 1 if employee e is assigned to work with skill s at the period $p \in \text{Periods}(d)$, 0 otherwise.

$$\sum_{s \in \text{Skills}(e)} X(e, s, p) \leq 1, \forall e, \forall p. \quad (5)$$

The needs in skill s of each daily period p , designated by $\text{ND}(s, p)$, are covered if

$$\sum_{e \in \text{Employees}} X(e, s, p) \geq \text{ND}(s, p), \forall s, \forall p. \quad (6)$$

It is straight forward to link the variables X to those in capacity planning. If we define the auxiliary binary variables $U(e, p) = \sum_{s \in \text{Skills}(e)} X(e, s, p)$. They take the value 1 if e is working on period p and 0 otherwise.

$$\text{WD}(e, w) = \sum_{p \in \text{Periods}(w)} U(e, p), \forall e, \forall w \quad (7)$$

$$Y(e, s, w) = \sum_{p \in \text{Periods}(w)} X(e, s, p), \forall e, \forall s, \forall w. \quad (8)$$

At this stage the model can be used to produce schedules that cover stated requirements, but the employees may be required to work for periods scattered here and there and resting in between. Labor law stipulates that employees are paid a minimum duration of H_{\min} periods on any day. To produce compact and cost-effective schedules, we use patterns, similar to that proposed in [8].

Patterns on a daily horizon

A pattern n in the set of **Patterns** is defined by the subset of periods that it covers, i.e. $v(n, p) = 1$ if pattern n covers period p . Valid patterns are those that require employees to work on compact schedules, with adequate meal/short breaks. We define a supplementary decision variable $X'(e, n)$ taking the value 1 when employee e is assigned to pattern n ; the following equations hold:

$$\sum_{n \in \text{Patterns}} X'(e, n) = 1, \forall e. \quad (9)$$

$$\sum_{s \in \text{Skills}(e)} X(e, s, p) \leq \sum_{n \in \text{Patterns}} X'(e, n) v(n, p), \forall e, \forall p. \quad (10)$$

Equation (9) stipulates that each employee is assigned to one and only one pattern. For a given employee e and period p , if e is assigned to pattern n which covers p , then $\sum_{s \in \text{Skills}(e)} X(e, s, p) \leq 1$ and e may be assigned to a skill s or to rest. If pattern n does not cover period p , the sum is 0 and $\sum_{s \in \text{Skills}(e)} X(e, s, p) \leq 0$, i.e. e must be at rest.

Designating the cost of assigning employee e to pattern n by $c(e, n)$, the total cost is

$$\sum_{e \in \text{Employees}} \sum_{n \in \text{Patterns}} X'(e, n) c(e, n) \quad (11)$$

Patterns render the schedule less flexible in assigning individual periods, although they interpret regulations such as minimum and maximum work durations (at the daily horizon in this case), acceptable break windows, etc. It allows valid solutions to be found rapidly but there may be over capacity on some periods. Without the equations (9) to (11), the model is limited to small instances with less than 15 employees, 3 skills and 44 periods. The implicit short-term scheduling method can also produce solutions quickly, see [11], [3], [5], etc. It reasons on the number of assignment changes instead of the number of employees on the job: the resulting model cannot be directly related to the capacity planning model.

Patterns on a weekly horizon

We need to handle the sequence of patterns on successive days so as to respect minimal rest between them. For example, an employee finishing at 11 pm would not take the morning shift starting at 7 am the following day. Instead of the variable n , we have $n_d \in \text{Patterns}$, where $d \in \{1, 7\}$ in the equations (9) to (11).

Define a weekly pattern m by the Boolean variable $u(m, n, d) = 1$ if and only if n is the d^{th} daily pattern of the valid weekly pattern m . Each employee is assigned to one and only one weekly pattern per week.

$$\sum_{m \in \text{Weekly Patterns}} X''(e, m) = 1, \forall e. \quad (12)$$

$$X'(e, n_d) = \sum_{m \in W} \sum_{n \in \text{Patterns}} X''(e, m) u(m, n, d), \forall e, \forall n, \forall d \in \{1, 7\}. \quad (13)$$

To handle the weekly horizon, we replace $X'(e, n_d)$ by $X'(e, n, d)$.

4 Conclusions

In this paper, we discussed the concept of Strategic Employee Scheduling, its constituents and one possible implementation. Scheduling employees with a strategy: this is different from existing concepts in human resource management by the ability to handle extra-scheduling features such as team sizing, launching extra activities, or taking into account considerations outside the usual scheduling horizon. We aim to convince researchers that the world of human resource management is very rich and there are many aspects that must be taken into account, instead of the homogeneous resources first discussed 50 years ago.

To implement the planning and scheduling components, we proposed MIP models for capacity planning and detailed scheduling that can be directly related to each other (i.e. (8)). Building onto the pattern model of [8] published in April 2006, we see that patterns are well suited to planning at multiple horizons, since they implement sets of assignments of one level which may be manipulated at the next. We are currently in the process of validating the system and no computation results are available. It is not our aim to propose THE model for solving Strategic Employee Scheduling; we encourage researchers to look into the MBP described in § 1 and propose their models.

Acknowledgement: We wish to thank Dr. Troy Daniels of BaeSystems for helping us to translate our first logical conditions into linear equations, at the Lp_Solve forum at http://groups.yahoo.com/group/lp_solve/.

We are also thankful to the reviewers who proposed many changes to improve this paper, and the second part was subsequently rewritten.

References

1. Bard J.F., Purnomo H.W., *Preference Scheduling for Nurses using Column Generation*, European Journal of Operational Research **164**: 510 – 534, 2005.
2. Barták R., Rudová H., *Modelling for Planning, Scheduling, and Timetabling Problems*, in the proceedings of the 20th Workshop of the UK Planning and Scheduling SIG, Edinburgh, UK, 2001. <http://planning.cis.strath.ac.uk/plansig/>
3. Bechtold S.E., Jacobs L.W. *The equivalence of general set-covering and implicit integer programming formulations for shift scheduling*, Naval Research Logistics **43**:233-249 1996.
4. Chan P., Zemmour T., Hiroux M., Weil G., *Multiple-level Models: an application to employee timetabling*, 5th International Conference on Practice and Theory of Automated Timetabling PATAT 2004 at Pittsburgh, USA. <http://mat.gsia.cmu.edu/patat04>
5. Chan P., Zemmour T., Hiroux M., Weil G., *Modelling Multi-skilled Workforces*, International Conf. OR2005 University of Bremen, Germany, Section on Scheduling and Project Management. <http://www.or2005.uni-bremen.de/download/finalschedule/program.pdf>
6. Cox T., Jr. *Towards the measurement of manufacturing flexibility*, Production Inventory Management Journal, First Quarter: 68 – 72, 1989.
7. Dantzig G.B. *A Comment on Edie's Traffic Delays at Toll Booths*. Operational Research **2**: 339 – 341, 1954.
8. Detienne B., Peridy L., Pinson E., Rivreau D. *Génération de coupes pour la planification d'agents*. Int. Conference Modélisation et Simulation MOSIM 2006, Maroc, 3-5/04/2006.
9. Glover F., McMillan C., *The General Employee Scheduling Problem: An integration of MS and AI*. Computers & Operations Research. **13** : 563 – 573, 1986.
10. Haselünne J.H. *Models and Algorithms for Ground Staff Scheduling on Airports*. Thesis defended at Technical University of Aachen, in collaboration with Dash Optimization, 2005.
11. Moondra S.L., *LP model for work force timetabling in banks*, Journal of Bank Resources, **7**: 299 – 301, 1976.
12. Mooney E., Davidson T., *Tour Scheduling with Skill Based Costs*, 5th Internat. Conf. on Practice Theory of Automated Timetabling, US, 2004. <http://mat.gsia.cmu.edu/PATAT04/>
13. Muscettola N., Pollack M.E., Tutorial on Temporal and Resource Reasoning for Planning, Scheduling and Execution, ICAPS 2005. <http://icaps05.icaps-conference.org>
14. Rekik M., Cordeau J.F., Soumis F., *Implicit Shift Scheduling with Multiple Breaks and Pre and post Break Restrictions*, Les Cahiers du GERAD numbers G-2002-32 and G-2005-15. <http://www.gerad.ca/fr/publications/>

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. Given a planning horizon with a fixed number of periods the objective is to avoid situations, or at least to minimize them, when a student is enrolled in two exams that are scheduled for the same period. Ant colony approaches have been proven to be a powerful solution approach for various combinatorial optimization problems. In this paper a Max-Min and a ANTCOL approach will be presented. Its performance is compared with other approaches presented in the literature and with modified graph coloring algorithms.

Key words: scheduling, exam timetabling, ant colony algorithms, Max-Min approach, graph coloring

1 Introduction

The exam timetabling problem faces the problem of scheduling exams within a limited number of available periods. As students plan to write different exams, 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's 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 x 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 [1] a cost function that imposes penalties P_ω for a conflict of order ω , i.e. whenever one student has to write two exams scheduled within $\omega + 1$ consecutive periods. In the literature ω 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 can not be written within specific periods. References [2–4] 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.

To solve exam timetabling problems, several algorithms have recently been developed. In [1] Carter et al. applied some well known graph coloring heuristics which they combined with backtracking.

In recent time various heuristical approaches have been developed. Most of them use local search like tabu search, simulated annealing, great deluge or adaptive search methods [5, 6, 1, 7, 8, 2, 9–11]. A comprehensive survey on the literature on exam timetabling problems can be found in [4].

The aim of this paper is twofold: Originally, this research was motivated by the need for a software tool for solving a practical exam timetabling problem. As ant colony approaches have been proven to be a powerful tool for various combinatorial optimization problems (c.f. the survey in [12]), it is apparent to adapt this solution approach to the exam timetabling problem. In the literature different variants of ant colony approaches have been presented. We will compare some of these strategies with respect to their suitability for our problem.

This paper is organized as follows: In section 2 a detailed problem formulation will be presented. Section 3 will give an introduction into ant colony systems. The next sections will present a solution approach and test results for some benchmark problems that were taken from the literature. Finally, section 6 summarizes the results and suggests discussion for future work.

2 Problem formulation

Before stating the problem formally, we introduce some notation.

R index set of rooms

I index set of exams

T index set of periods

Ω 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_i number of students enrolled in exam i

P_ω penalty imposed if one student has to write two exams
within $\omega + 1$ periods

y_{it} binary variable equal to 1 if exam i is scheduled in period t
and 0 otherwise

p_{irt} 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:

$$\min \sum_{\omega \in \Omega} \sum_{i,j \in I, i \neq j} \sum_{t \in T, t > \omega} P_\omega c_{ij} y_{it} y_{j(t-\omega)} \quad (1)$$

s.t.

$$\sum_{t \in T} y_{it} = 1 \quad \forall i \in I \quad (2)$$

$$p_{irt} \leq y_{it} K_{rt} \quad \forall i \in I, \forall r \in R, \forall t \in T \quad (3)$$

$$\sum_{r \in R} \sum_{t \in T} p_{irt} = E_i \quad \forall i \in I \quad (4)$$

$$\sum_{i \in I} p_{irt} \leq K_{rt} \quad \forall r \in R, \forall t \in T \quad (5)$$

$$\sum_{t \in T} c_{ij} y_{it} y_{jt} = 0 \quad \forall i, j \in I, i \neq j \quad (6)$$

$$y_{it} \in \{0, 1\} \quad \forall i \in I, \forall t \in T \quad (7)$$

$$p_{irt} \in \mathbb{N}_0 \quad \forall i \in I, \forall r \in R, \forall t \in T \quad (8)$$

The objective function (1) balances out students' workload by minimizing the weighted sum of all conflicts. 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) assure 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 [13]. Therefore, solution approaches try to decompose the problem in order to solve it within a reasonable amount of time [14]. One way is to split up the problem into the two following subproblems, which can be solved sequentially:

Problem I: *Scheduling of exams*, i.e. assign exams to periods in order to balance out students' workload as pursued by the objective function (1). Instead of considering capacity constraints for the single rooms, only the total capacity of all available exam rooms within a period is considered. In the IP formulation stated above this can be accomplished by replacing the set of rooms by a artificial single room. For this problem a solution approach will be presented in the next sections.

Problem II: *Room planning*, i.e. distribute the exams of one period among the available examination rooms. Finding a feasible room plan is not difficult if the exams can take place in more than one room and if more than one exam can take place in one room at the same time, provided that the room capacity is not exceeded. If exams are split up into different rooms one could consider the campus layout and try to generate a room plan where these exams are only assigned to rooms not too far from each other in order to minimize walking distances. We will not consider this problem in the following.

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 Colorni, Dorigo and Maniezzo in the early nineties [15]. See [12] for an in depth introduction into 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 systems (AS). We will briefly describe the basic principle of AS algorithms by means of the TSP. This solution approach to the TSP will 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 the knowledge gathered by other ants in the past on how good solutions can be obtained is incorporated into the ant's 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 later referred to as η_{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 τ which is initialized with 0 for all arcs (i, j) . 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 τ_{ij} associated to arc (i, j) is intended to represent the learned desirability of choosing node j when in node i . Consequently, arcs belonging to good solutions receive a high amount of pheromone.

AS algorithms combine these two strategies. The probability that an ant ν located in node i chooses the next node j is determined by the following formula:

$$p_{ij}^\nu = \begin{cases} \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{k \in N_i^\nu} (\tau_{ik})^\alpha (\eta_{ik})^\beta} & \text{if } j \in N_i^\nu \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

α and β are given weighting factors and N_i^ν is the set of nodes that have not yet been visited by ant ν currently located in node i .

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, like e.g. ant colony systems (ACS) or Max-Min ant systems (MMAS), which obtained much better results than AS (c.f. [12]). In particular, MMAS, which was first proposed by Stützle and Hoos [16], generated significantly better solutions for the TSP than AS. Socha et al. [17] compared the MMAS variant with ACS and found out that MMAS outperformed the ACS approach for the considered timetabling problem.

The main modification of MMAS are related to the way how the matrix τ is initialized and how pheromone values are updated. Additionally, MMAS uses local search to improve the solutions found by the ants. Details will be discussed in the next section.

As far as the author is aware, ant colony algorithms to scheduling problems have only been applied by Colorni et al. [15] and by Socha et al. [17]. The former article focuses on the job shop scheduling problem, the latter one on the timetabling problems for university classes, which are slightly different from the exam timetabling problem considered here. Finally, Costa and Hertz [18] used an ant colony approach to solve assignment type problems, in particular graph coloring problems. Recently, Dowsland and Thomson as well as Vesel and Zerovnik modified and improved in [19, 20] this graph coloring algorithm with respect to the examination scheduling problem.

4 An ant algorithm for the exam scheduling problem

4.1 General modifications for the exam timetabling problem

Like in AS, the solution approach consists of n cycles. In each of these cycles first each of the m ants constructs a feasible solution using therefore 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 how this basic solution approach can be adapted to the exam timetabling problem.

- At each stage of the construction process in the AS approach of Costa and Hertz [18] called ANTCOL the ant chooses first a node i and then a feasible color according to a probability distribution equivalent to (9). The matrix τ 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 [12] 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.

- In Socha et al. [17] 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 τ_{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.

For the exam timetabling problem the way the information in matrix τ is used in both approaches is not 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 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 we implemented a two step approach.

Step I: Determine the sequence according to the exams is scheduled. Like 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 this ant constructs the tour corresponds to the sequence in which the exams are scheduled.

Step II: Find the most suitable period for an exam which should be scheduled. Therefore, all admissible periods are evaluated according to the given penalty function.

Following this two step approach probabilities p_{ij}^ν for choosing the next node j that has to be scheduled are computed according to (9). Pheromone values τ_{ij}

along the ants' paths are updated by the inverse of the objective function value. For the heuristic value η_{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 [1]. The value η_{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 (c.f. [16]):

- Pheromone trails are only updated by the ant that generated the best solution in a cycle. The corresponding values τ_{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 τ_{ij} is updated by $(1 - \rho)\tau_{ij}$. $\rho \in [0, 1]$ 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 τ_{ij} is set to τ_{min} or τ_{max} , respectively. The rationale behind this are 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 τ_{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 list has been proven to reduce required computational times as well as to improve solution quality at the same time (c.f. [12]). 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 in case all exams in this list have already been scheduled, the remaining exams are 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 twelve practical examination problems can be found on the site of Carter (c.f. [21]). Table 1 summarizes some characteristics of these problems. 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_ω is the penalty for the constrain violation of order ω . 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.

Table 1. Test cases from Carter et al. [1, 21, 22]

test case	# exams	# students	# student exams	problem density	# periods
car-f-92	543	18419	55522	13.8 %	32
car-s-91	682	16925	56877	12.8 %	35
ear-f-83	190	1125	8109	26.7 %	24
hec-s-92	81	2823	10632	42.0 %	18
kfu-s-93	461	5349	25113	5.6 %	20
lse-f-91	381	2726	10918	6.3 %	18
pur-s-93	2419	30032	120681	2.9 %	43
rye-f-92	486	11483	45051	7.5 %	23
sta-f-83	139	611	5751	14.4 %	13
tre-s-92	261	4360	14901	5.8 %	23
uta-s-92	622	21267	58979	12.6 %	35
ute-s-92	184	2750	11793	8.5 %	10
yor-f-83	181	941	6034	28.9 %	21

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 ρ was set to 0.3. Like in [16] it turned out that this parameter is quite robust, i.e. the parameter ρ 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 τ_{max} and τ_{min} as proposed in [16].
- Different values for the weighting factors α and β were tested. It turned out that the approach performed best when α was set to one and β was chosen high. Best results were obtained for β equal to 24. But the difference was on the average less than one percent when β was bigger than eight. A high β forces 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 η_{ij} is the inverse of the saturation degree as explained in section 4.1. Thus, a high β 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. Whereas, for small values of β , 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 twenty 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. For each approach the minimal objective function value and the average result after twenty test runs are given. 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.

Thus, one could ask how much the ants contribute to the solution or if solutions of the same quality could also be achieved by applying only the hill climber on a random starting solution. Therefore a third version of the MMAS-ET approach was implemented where each ant constructs an exam timetable without interacting with the other ants, i.e. the matrix τ is not updated at all. This

approach can be seen as a randomized greedy heuristic. As in MMAS-ET with 50 ants and 50 cycles 2500 exam timetables were generated. The best solutions of this approach are displayed in the last column of table 2.

As the MMAS-ET approach without ants generates the worst solutions it is obvious, that the ant colony has a positive impact on the diversification of the solution space, i.e. the ants guide the search process into promising regions of the solution space where the hill climber can find good 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 twenty iterations was improved nor were better solutions found during the twenty iterations.

Table 2. Results for three different variants of the MMAS-ET approach for twenty test runs

test case	MMAS-ET		MMAS-ET		MMAS-ET	
	best	avg.	best	avg.	best	avg.
car-f-92	4.8	4.9	7.8	8.0	10.9	13.3
car-s-91	5.7	5.9	9.3	9.5	11.9	13.9
ear-f-83	36.8	38.6	50.4	53.0	49.5	62.4
hec-s-92	11.3	11.5	14.8	15.8	11.6	15.5
kfu-s-93	15.0	15.5	23.9	24.6	19.5	22.0
lse-f-91	12.1	12.7	19.3	19.8	16.7	25.4
pur-s-93	5.4	5.6	12.2	12.5	11.7	14.6
rye-s-93	10.2	10.4	18.0	18.7	12.2	14.2
sta-f-83	157.2	157.5	160.6	161.9	157.3	157.7
tre-s-92	8.8	9.1	12.4	12.8	9.2	13.1
uta-s-92	3.8	3.8	6.2	6.3	8.2	9.9
ute-s-92	27.7	28.6	33.6	34.5	27.7	30.1
yor-f-83	39.6	40.3	50.5	51.3	62.9	73.0

5.4 Comparison with other exam timetabling approaches

The proposed MMAS-ET approach was compared with the following approaches:

- LD, SD, LDW and LE: Carter et al. compared in [1] four different priority rules largest degree (LD), saturation degree (SD), largest weighted degree (LWD) and largest enrollment (LE).
- Wal: Tabu search approach with longer-term memory proposed by White et al. in [11].
- GS: Tabu search approach proposed by Di Gaspero and Schaerf in [2].
- Cal: Local search approach of Caramia et al. [6].
- BN: Great deluge local search approach developed by Burke and Newall [5].

- Mal: Simulated annealing approach of Merlot et al. [9].
- Ga: Multi-neighborhood search approach presented by Di Gaspero [8]
- PS: Tabu search approach of Paquete and Stützle [10].
- CT: Randomized adaptive search algorithm of Casey and Thomson [7].

The results of the benchmarks are taken from the literature [11] and from the internet (c.f. the timetabling database at the University of Melbourne [22]). Table 3 displays the best solution and the average solution achieved when each test case was solved twenty times. The results of table 3 can be summarized as follows:

Table 3. Best (b.) and average (a.) solution after twenty test runs for the benchmark test cases from Carter et al.[1, 21, 22]

test case	LD	SD	LWD	LE	Wal	GS	Cal	BN	Mal	Ga	PS	CT	MMAS -ET
car b.	7.6	6.6	6.6	6.2	4.6	5.2	6.0	4.0	4.3	-	-	4.4	4.8
-f-92 a.	7.6	6.6	6.6	6.2	4.7	5.6	6.0	4.1	4.4	-	-	4.7	4.9
car b.	7.9	7.1	7.4	7.6	5.7	6.2	6.6	4.6	5.1	5.7	-	5.4	5.7
-s-91 a.	7.9	7.1	7.4	7.6	5.8	6.5	6.6	4.7	5.2	5.8	-	5.6	5.9
ear b.	36.4	46.5	37.3	42.3	45.8	45.7	29.3	36.1	35.1	39.4	40.5	34.8	36.8
-f-83 a.	36.4	46.5	37.3	42.3	46.4	46.7	29.3	37.1	35.4	43.9	45.8	35.0	38.6
hec b.	10.8	12.7	15.8	15.9	12.9	12.4	9.2	11.3	10.6	10.9	10.8	10.8	11.3
-s-92 a.	10.8	12.7	15.8	15.9	13.4	12.6	9.2	11.5	10.7	11.4	12.0	10.9	11.5
kfu b.	14.0	15.9	22.1	20.8	17.1	18.0	13.8	13.7	13.5	-	16.5	14.1	15.0
-s-93 a.	14.0	15.9	22.1	20.8	17.8	19.5	13.8	13.9	14.0	-	18.3	14.3	15.5
lse b.	12.0	12.9	13.1	10.5	14.7	15.5	9.6	10.6	10.5	12.6	13.2	14.7	12.1
-f-91 a.	12.0	12.9	13.1	10.5	14.8	15.9	9.6	10.8	11.0	13.0	15.5	15.0	12.7
pur b.	4.4	4.1	5.0	3.9	-	-	3.7	-	-	-	-	-	5.4
-s-93 a.	4.4	4.1	5.0	3.9	-	-	3.7	-	-	-	-	-	5.6
rye b.	7.3	7.4	10.0	7.7	11.6	-	6.8	-	8.4	-	-	-	10.2
-s-93 a.	7.3	7.4	10.0	7.7	11.7	-	6.8	-	8.7	-	-	-	10.4
sta b.	162.9	165.7	161.5	161.5	158.0	161.0	158.2	168.3	157.3	157.4	158.1	134.9	157.2
-f-83 a.	162.9	165.7	161.5	161.5	158.0	167.0	158.2	168.7	157.4	157.7	159.3	135.1	157.5
tre b.	11.0	10.4	9.9	9.6	8.9	10.0	9.4	8.2	8.4	-	9.3	8.7	8.8
-s-92 a.	11.0	10.4	9.9	9.6	9.2	10.5	9.4	8.4	8.6	-	10.2	8.8	9.1
uta b.	4.5	3.5	5.3	4.3	4.4	4.2	3.5	3.2	3.5	4.1	-	-	3.8
-s-92 a.	4.5	3.5	5.3	4.3	4.5	4.5	3.5	3.2	3.6	4.3	-	-	3.8
ute b.	38.3	31.5	26.7	25.8	29.0	29.9	24.4	25.5	25.1	-	27.8	25.4	27.7
-s-92 a.	38.3	31.5	26.7	25.8	29.1	31.3	24.4	25.8	25.2	-	29.4	25.5	28.6
yor b.	49.9	44.8	41.7	45.1	42.3	41.0	36.2	36.8	37.4	39.7	38.9	37.5	39.6
-f-83 a.	49.9	44.8	41.7	45.1	42.5	42.1	36.2	37.3	37.9	40.6	41.7	38.1	40.3

Although, the MMAS-ET approach does not generate outstanding results its performance is comparable with other approaches. Beside the graph coloring

heuristics of Carter et al. it also finds better solutions than the Wal, the GS, the PS, the Ga and the CT approach for most test cases.

In addition, it is striking that no approach outperforms all other approaches for all test cases. Thus, there are some test cases where MMAS-ET outperforms the approaches Cal, BN and CT, although one must confirm that these three approaches generate better solutions for most of the test cases. For example, MMAS-ET found better solutions than the Ca approach in four out of the 13 test cases, i.e. for the test cases car-f-92, car-s-91, sta-f-83 and tre-s-92.

5.5 Comparison with the approach of Costa and Hertz

Finally, the results of MMAS-ET were compared with a modified version of the ANTCOL algorithm of Costa and Hertz [18], which originally was 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 [18]. The objective function was modified in order to consider conflicts of higher order too. Test runs were carried out to adjust the parameters appropriately. The parameter α was set to 1, β to 35. ρ was set equal to 0.3. Again, each test case was solved twenty times.

Table 4 shows the results for the thirteen 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 (c.f. [12, 16]).

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 the average better solutions than the MMAS-ET approach and

Like the MMAS-ET approach the ANTCOL-ET approach in particular improves the test cases that already achieved the best solutions. For example, it again outperformed the approach of Caramia et al. in the test cases car-f-92, car-s-91, sta-f-83 and tre-s-92. White et al. argued in [11] that these test cases seem to be in a way easier.

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 can conclude that ANTCOL-ET takes more time but gets a better solution quality than MMAS-ET. Please note that the same stopping criteria was used for both algorithms, namely, 2500 solutions.

6 Conclusion

In this paper different strategies for solving exam timetabling problems were tested. Ant colony approaches are capable of solving large real world exam

Table 4. Comparison between different ant colony approaches.

test case		MMAS-ET	ANTCOL-ET	ANTCOL-ET
		without hill climber with hill climber		
car-f-92	best	4.8	4.5	4.3
	avg.	4.9	4.6	4.4
car-s-91	best	5.7	5.3	5.2
	avg.	5.9	5.4	5.2
ear-f-83	best	36.8	40.3	36.8
	avg.	38.6	41.4	38.3
hec-s-92	best	11.3	12.2	11.1
	avg.	11.5	12.6	11.4
kfu-s-93	best	15.0	15.4	14.5
	avg.	15.5	15.8	14.9
lse-f-91	best	12.1	11.9	11.3
	avg.	12.7	12.2	11.7
pur-s-93	best	5.4	4.8	4.6
	avg.	5.6	4.9	4.6
rye-s-93	best	10.2	10.2	9.8
	avg.	10.4	10.7	10.0
sta-f-83	best	157.2	158.2	157.3
	avg.	157.5	159.3	157.5
tre-s-92	best	8.8	8.8	8.6
	avg.	9.1	9.0	8.7
uta-s-92	best	3.8	3.6	3.5
	avg.	3.8	3.7	3.5
ute-s-92	best	27.7	28.9	26.4
	avg.	28.6	29.4	27.0
yor-f-83	best	39.6	42.2	39.4
	avg.	40.3	43.7	40.4

timetabling problems. The implemented algorithms generated comparable results like other high performance algorithms from the literature.

Unlike for other combinatorial optimization problems like the TSP or the QAP for the exam timetabling problem the MMAS approach did not outperform the simpler AS strategy. Of course, it goes without saying but proper adjusting parameters can improve the performance of an algorithm considerably.

A self-evident extension would be to incorporate additional constraints and requirements like e.g. scarce room resources or precedence constraints between exams.

References

1. M.W. Carter, G. Laporte, and S.Y. Lee. Examination timetabling algorithmic strategies and applications. *Journal of the Operational Research Society*, 47:373–383, 1996.

2. L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. *Lecture Notes in Computer Science*, 2079:104–117, 2001.
3. J.P. Boufflet and S. Nègere. Three methods used to solve an examination timetable problem. *Lecture Notes in Computer Science*, 1153:327–344, 1996.
4. M.W. Carter and G. Laporte. Recent developments in practical examination timetabling. *Lecture Notes in Computer Science*, 1153:3–21, 1996.
5. E.K. Burke and J. Newall. Enhancing timetable solutions with local search methods. *Lecture Notes in Computer Science*, 2740:195–206, 2003.
6. M. Caramia, P. Dell'Olmo, and G.F. Italiano. New algorithms for examination timetabling. *Lecture Notes in Computer Science*, 982:230–241, 2001.
7. S. Casey and J. Thompson. Grasping the examination scheduling problem. *Lecture Notes in Computer Science*, 2740:233–244, 2003.
8. L. Di Gaspero. Recolour, shake and kick: A recipe for the examination timetabling problem. In *Proceedings of the fourth international conference on the practice and theory of automated timetabling*, pages 404–407, Gent, Belgium, August 2002.
9. L.T.G. Merlot, N. Boland, B.D. Hughes, and P.J. Stuckey. New benchmarks for examination timetabling. Testproblem Database, <http://www.or.ms.unimelb.edu.au/timetabling.html>.
10. L. Paquete and T. Stuetzle. Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. In *Proceedings of the fourth international conference on the practice and theory of automated timetabling*, pages 413–420, Gent, Belgium, August 2002.
11. G.M. White, B.S. Xie, and S. Zonjic. Using tabu search with long-term memory and relaxation to create examination timetables. *European Journal of Operational Research*, 153:80–91, 2004.
12. M. Dorigo, G. Di Caro, and L.M. Gambarella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.
13. M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Company, New York, 1979.
14. V. Lofti and R. Cerveny. A final-exam scheduling package. *Journal of the Operational Research Society*.
15. A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the first european conference on artificial life*, pages 134–142, Amsterdam, 1992. Elsevier Science Publishers.
16. T. Stuetzle and H.H. Hoos. Max-min ant systems. *Future Generation Computer System*, 16:889–914, 2000.
17. K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to state-of-the-art. In *Proceedings of 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP'2003)*, pages 334 – 345, Essex, UK, April 2003.
18. D. Costa and A. Hertz. Ants can color graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.
19. K. Dowsland and J. Thompson. Ant colony optimisation for the examination scheduling problem. *Journal of the Operational Research Society*, 56:426–439, 2005.
20. A. Vesel and J. Zerovnik. How well can ants color graphs? *Journal of Computing and Information Technology - CIT*, 8:131–136, 2000.
21. <ftp://ie.utoronto.ca/pub/carter/testprob>.
22. <http://www.or.ms.unimelb.edu.au/timetabling/ttxp2.html>.

The KTS High School Timetabling System

Jeffrey H. Kingston

School of Information Technologies
The University of Sydney, NSW 2006, Australia
<http://www.it.usyd.edu.au/~jeff>
jeff@it.usyd.edu.au

Abstract. KTS is a web-based software system for solving high school timetabling problems, freely accessible on the Internet. This paper describes 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

Research into automated timetabling has had many successes in recent years. Examination timetabling and university course timetabling have yielded to meta-heuristic methods, as the proceedings of recent PATAT conferences [1, 2] show.

High school timetabling has had much less success [3], probably because it is dominated by hard constraints, to which meta-heuristics seem less well suited [8]. There may also be non-technical reasons, such as fewer researchers in the field and less ready access to data.

KTS is a web server for high school timetabling created by the author. Its web interface puts the system on the desk of the 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 [6].

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 considered first 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*. A resource selection 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; 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 [4]. The user may upload and download KTS instance files, although there is no strong motive for doing so.

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 **Update Meeting**

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).

3 User interface

The KTS system is not distributed to users for installation on their own systems. Instead, there is a unique copy running on a server 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 at any time; 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.

Bankstown Girls High School, 1998	Satisfactory		Unsatisfactory		Total	
	No.	%	No.	%	No.	%
Assignments to time selections						
Required blocks	<u>147</u>	96.7	<u>5</u>	3.3	<u>152</u>	100.0
Suggested blocks	<u>152</u>	100.0	<u>0</u>	0.0	<u>152</u>	100.0
At most 1 Undesirable time	<u>152</u>	100.0	<u>0</u>	0.0	<u>152</u>	100.0
Even spread through Day 1 .. Day 5	<u>120</u>	78.9	<u>32</u>	21.1	<u>152</u>	100.0
Assignments to resource selections						
Rooms	<u>229</u>	97.9	<u>5</u>	2.1	<u>234</u>	100.0
Students	<u>316</u>	100.0	<u>0</u>	0.0	<u>316</u>	100.0
Teachers	<u>408</u>	92.3	<u>34</u>	7.7	<u>442</u>	100.0
Soft workload limits						
Teachers	<u>266</u>	95.0	<u>14</u>	5.0	<u>280</u>	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.

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 three 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.

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*. The basic approach appeared in an earlier paper by the author [5], but the present work describes a completely rewritten solver, with more and better results.

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.

Hall Set 2

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
Total tixels demanded				3

To satisfy this demand, the following tixels of supply are available:

Resource	Time(s)	Tixels
Diamond	Mon5	1
Leeon	Mon5	1
Total tixels supplied		2

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.

4.1 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 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 pre-assigned 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.

	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.

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.

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

Time 0	Time 1	Time 2	Time 3	Time 4	Time 5
8CKOAS-Maths					8C-History 8K-History 8O-History 8A-History 8S-History
8C-English					8C-Music
8K-English				8K-Music	8K-English
8O-English		8O-Music		8O-English	
8A-English		8A-Music	8A-English		
8S-English	8S-Music	8S-English			

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.

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 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 preassigned 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 [7]. 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.

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.

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.

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.

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 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.

	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	8C-Sci	7K-Science	8A-Science	7A-Sci	8A-Sci													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					8K-Sci	12-2-GeneralScience-1					10-Science3														
Saule	0	8O-Science						12-2-Biology					10-Science5													12-3-Biology	
Smith	1			9-Science-3	7S-Sci					7S-Science		8O-Sci	10-Science4												11-5-Physics		
Unassigned												7K-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																			
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.

Room assignment is not difficult. The solver assigns each time block of each meeting, 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.

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.

Table 1. The six instances tested, showing the number of times, resources, and meetings in each.

<i>Instance</i>	<i>Times</i>	<i>Student Groups</i>	<i>Teachers</i>	<i>Rooms</i>	<i>Meetings</i>
<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

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

This section analyses the performance of the solver on six instances taken from three high schools in Sydney, Australia. Statistical descriptions of these 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.

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

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 0.0 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.

<i>Instance</i>	<i>Tile construction</i>	<i>Time assignment</i>	<i>Resource assignment</i>	<i>Total</i>
<i>bghs93</i>	0.0	3.0	3.0	6.0
<i>bghs95</i>	0.0	1.0	7.0	8.0
<i>bghs98</i>	0.0	1.0	6.0	7.0
<i>tes98</i>	1.0	1.0	0.0	2.0
<i>tes99</i>	0.0	1.0	0.0	1.0
<i>sahs96</i>	1.0	31.0	0.0	32.0

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.

<i>Instance</i>	<i>Block structure</i>	<i>Spread</i>	<i>Undesirable times</i>
<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%)

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.

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)
bghs93	0 (0.0%)	7 (3.1%)	15 (1.2%)	15 (1.2%)
bghs95	0 (0.0%)	6 (2.9%)	9 (0.7%)	9 (0.7%)
bghs98	0 (0.0%)	5 (2.1%)	7 (0.5%)	7 (0.5%)
tes98	2 (2.2%)	5 (5.5%)	7 (1.5%)	7 (1.5%)
tes99	0 (0.0%)	5 (3.7%)	7 (1.3%)	7 (1.3%)
sahs96	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)
bghs93	3 (0.7%)	7 (1.5%)	5 (0.3%)	9 (0.6%)
bghs95	17 (3.7%)	15 (3.3%)	7 (0.5%)	27 (2.0%)
bghs98	24 (5.4%)	10 (2.3%)	8 (0.5%)	17 (1.2%)
tes98	7 (3.8%)	13 (7.1%)	14 (3.0%)	14 (3.0%)
tes99	2 (1.1%)	9 (5.1%)	9 (1.7%)	9 (1.7%)
sahs96	0 (0.0%)	27 (11.2%)	47 (3.1%)	47 (3.1%)

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 Conclusion

This paper has presented KTS, a freely accessible web-based system for high school timetabling which produces good timetables in a few seconds.

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.

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 [7].

In parallel with these efforts, the KTS system will be promoted to Australian high schools. At the time of writing, 60 accounts have been created, but only a few are active. More users will bring a larger and more diverse set of test instances, which should lead to further progress.

References

1. Edmund Burke and Wilhelm Erben (eds.): Practice and Theory of Automated Timetabling III (PATAT2000, Konstanz, Germany, August 2000, Selected Papers). Springer Lecture Notes in Computer Science 2079, 2001
2. Edmund Burke and Patrick de Causmaecker (eds.): Practice and Theory of Automated Timetabling IV (PATAT2002, Gent, Belgium, August 2002, Selected Papers). Springer Lecture Notes in Computer Science 2740, 2003
3. M. W. Carter and Gilbert Laporte: Recent developments in practical course timetabling. Practice and Theory of Automated Timetabling II (Second International Conference, PATAT'97, University of Toronto, August 1997, Selected Papers), Springer Lecture Notes in Computer Science 1408, pages 3-19, 1008
4. Tim B. Cooper and Jeffrey H. Kingston: The solution of real instances of the timetabling problem. *The Computer Journal* **36**, pages 645–653, 1993
5. Jeffrey H. Kingston: A tiling algorithm for high school timetabling. In Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, PA, pages 233-249, August 2004
6. Jeffrey H. Kingston: The KTS high school timetabling web site (Version 1.3), October 2005. <http://www.it.usyd.edu.au/~jeff>
7. Jeffrey H. Kingston: Hierarchical timetable construction. To appear in Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, Czech Republic, August 2006
8. Peter Ross, Emma Hart, and Dave Corne: Some observations about GA-based exam timetabling. Practice and Theory of Automated Timetabling II (Second International Conference, PATAT'97, University of Toronto, August 1997, Selected Papers), pages 115-129, 1998

Hierarchical Timetable Construction

Jeffrey H. Kingston

School of Information Technologies
The University of Sydney, NSW 2006, Australia
<http://www.it.usyd.edu.au/~jeff>
jeff@it.usyd.edu.au

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 [3–7].

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, where teachers are assigned as well as rooms. Teacher assignment is the main area where the author's previous work in high school timetabling [8, 10] 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, and a data structure, the *layer tree*, which represents these expressions and efficiently supports the basic assignment and deassignment operations on which most timetable construction algorithms are built. This author's KTS timetabling system [11, 12], a free, public web site for high school timetabling, uses layer trees. They 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.

Our focus is on the efficient implementation of the basic assignment and deassignment operations, 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. KTS typically produces a good timetable in about ten seconds [12], showing that layer trees can support practical timetabling.

Much of this paper is concerned with constraint propagation, but the emphasis here is on the efficient implementation of a particular set of constraints relevant to timetabling, rather than the use of a general-purpose constraint programming system to solve timetabling problems. Some of the algorithms used here, for example weighted bipartite matching, do not seem to be available in any existing constraint programming system [2, 9], although some recent research into the *all_different* constraint [13], which implements unweighted bipartite matching, is a step in that direction.

The algorithms used here have appeared in previous timetabling work by the author and others [8, 10, 14]. 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 $t\text{dom}(v)$, and its initial value, specified when the variable is created, will be denoted $t\text{dom}_0(v)$. For example, if v may be assigned any time, then $t\text{dom}_0(v) = T$; if v is *preassigned* to a specific time t , $t\text{dom}_0(v) = \{t\}$. 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 unification of one time variable, v , to another, w , with the meaning that v 's value is constrained to be equal to w 's. Unifying two variables expresses the idea that two meetings are to occur simultaneously, without having to say when.

Thus, our system offers two basic operations: unifying a variable v to one other variable w , and removing the unification of v to w . A variable may be unified to at most one other variable at any moment; but that other variable is free to be unified to a third variable (or not), and many variables may be unified 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 those 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.

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

$$w_1 w_2 \dots w_k : e$$

where $w_1 w_2 \dots w_k$ 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 unified 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 a timetable expression. The expression e may be timetabled into $w_1 w_2 \dots w_k$ 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

$$w_1 w_2 \dots w_n : e$$

where $t\text{dom}_0(w_i) = \{t_i\}$ for all i . Although the operation of assigning a particular time t_i to a variable v is not offered, unifying v to w_i is effectively the same thing.

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.

An example of a small timetable expression appears in Figure 1.

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 do occur, they are beyond the scope of this paper, and we will now 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

$$w_1 w_2 \dots w_n : (e_{11} e_{12} \dots e_{1k_1} + \dots + e_{m1} e_{m2} \dots e_{mk_m})$$

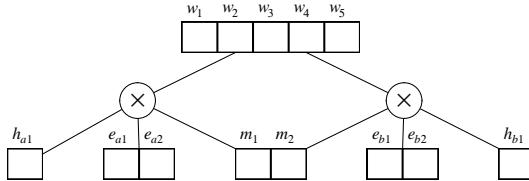
called a *restricted sum of products*. Here m may be 0, in which case the expression just denotes a sequence of variables $w_1 w_2 \dots w_n$. Each e_{ij} is a restricted sum of

	t_1	t_2	t_3	t_4	t_5
7A	7AB-Mathematics		7A-Hist	7A-English	
7B			7B-English		7B-Hist

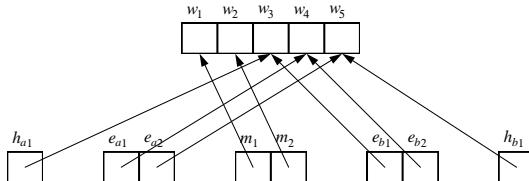
(a) A small timetable, or *tile*, occupying two student groups (7A and 7B) for five times t_1, t_2, t_3, t_4 , and t_5 .

$$w_1 w_2 w_3 w_4 w_5 : m_1 m_2 h_{a1} e_{a1} e_{a2} + m_1 m_2 e_{b1} e_{b2} h_{b1}$$

(b) A timetable expression for which (a) is a solution. Here $w_1 w_2 w_3 w_4 w_5$ represent the five available times, h_{a1} represents 7A-Hist, $e_{a1} e_{a2}$ represents 7A-English, and so on; $m_1 m_2$, representing 7AB-Mathematics, lies in two subexpressions.



(c) A layer tree corresponding to (b). Variables are shown as labelled boxes; \times nodes are shown as concatenations of their variables.



(d) The layer tree of (c), showing unifications representing the timetable of (a). The \times nodes have been omitted for clarity.

Fig. 1. Timetables, timetable expressions, layer trees, and unification.

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 unify 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 unifications and de-unifications 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 unifications 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 unification with a variable w in the $+$ node two levels above it. Each such unification is represented by a pointer in v to w (Figure 1d). Eventually, when all these variables are unified in this way, every variable may be said to have been assigned a time, obtainable by following the chain of pointers to its end.

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 1a; 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 unified to without violating any time constraints.

Since each variable is unified to at most one other variable at any moment, the unifications form a directed forest with edges pointing towards the roots. The current unification of a variable v will be denoted $p(v)$ ('parent of v ') when present, and the variable at the root of the tree of unifications 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 unified to other variables.

Recall that each time variable v has its *initial domain* $t\text{dom}_0(v)$ of times that it may be assigned initially, and its *current domain* $t\text{dom}(v)$ of times that it may be assigned to at the current moment. We require

$$t\text{dom}(v) \subseteq t\text{dom}_0(v)$$

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 unified 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 unified 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 unified and deunified:

$$tdom(v) = tdom_0(r(v))$$

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 unified.

When a variable v is unified to another variable w , the variable domains of all variables concatenated with v need to be reduced by removing w , since unifying any of them with w would violate the constraint that concatenated variables must be assigned distinct times. An efficient method of doing this is as follows.

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 in the parent of that \times node form another layer, which we call $p(L)$. The variables of L must be unified 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 unified directly to w_i . Parent layer sets must be maintained dynamically as unifications 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 unified to w . The set operations may be implemented efficiently using bit vectors.

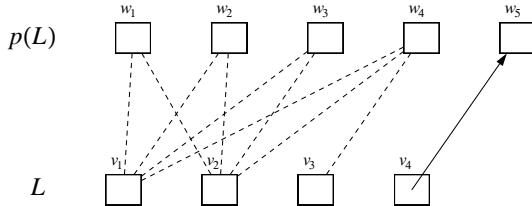


Fig. 2. An example of an unweighted bipartite matching graph between the variables of a child layer L and its parent layer $p(L)$, shown as dashed edges. One unification is already present, from v_4 to w_5 , ensuring that $L \in pl(w_5)$ and thus excluding w_5 from $vdom(v)$ for all other $v \in L$. This particular matching could arise when v_3 and w_4 are preassigned to the same time ($tdom(v_3) = tdom(w_4) = \{t_i\}$ 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 unify v_1 to w_4 .

Given current values of $vdom(v)$ for all variables v in some layer L , the next question is whether it is possible to unify all the currently un-unified variables of L to variables in $p(L)$. Since the unifications must be to distinct variables, this is an unweighted bipartite matching problem between the currently un-unified variables of L and the variables of $p(L)$, with edges defined by the current values of the domains $vdom(v)$ of the currently un-unified variables of L (Figure 2). We will see in the next section that there are reasons for preferring some unifications 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 meeting requests one preassigned student group resource, one teacher 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 is 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 the 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* [8], 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 exam-

ple, 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 $7C$, 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. The variables of the root layer, for example, have chunks that express resource unavailability: if 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 unified to that root variable directly or indirectly are accumulated (since the unifications 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 a unification 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 unified to v , directly or indirectly) onto $r(w)$'s matching graph; when a de-unification 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 unmatchable 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 un-unified variables of a layer L and the variables of its parent layer $p(L)$. For each un-unified 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 unified to w , calculated by matching the chunks of v 's subtree and $r(w)$'s subtree together without actually making the unification. A maximum matching between 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 un-unified variables of L are unified to variables of $p(L)$. This model has been called *weighted meta-matching* in [10], 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 [1]. Fortunately the graphs are small, since the number of nodes per layer is at most the number of times in the week (typically about 40), 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 matching contains all the parent layer blocks as supply nodes, and no child layer demand blocks. The child blocks are introduced into the matching one by one in decreasing width order. If a child block fails to match, a series of repair operations is tried on the parent blocks: larger blocks are split, variables not yet in any block are merged into blocks, and so on. For each type of repair, all possible repairs of that type are tried, and the one which produces a blocked matching of minimum cost is accepted; or if none of them succeed in producing a matching which touches every demand block, the algorithm proceeds to the next, less desirable, kind of repair. As a last resort, one demand block (usually the one just introduced) is dropped and replaced by its variables.

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 unifies the double to the first two times on Monday; it unifies 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 unifications 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 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.

Future work will try to add more features to the layer tree without compromising 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. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, 1993
2. Krzysztof R. Apt: Principles of Constraint Programming. Cambridge University Press, 2003
3. Edmund Burke and Peter Ross (eds.): Practice and Theory of Automated Timetabling (First International Conference, PATAT'95, Edinburgh, August 1995). Springer Lecture Notes in Computer Science 1153, 1995
4. Edmund Burke and Michael Carter (eds.): Practice and Theory of Automated Timetabling II (Second International Conference, PATAT'97, University of Toronto, August 1997, Selected Papers. Springer Lecture Notes in Computer Science 1408, 1998
5. Edmund Burke and Wilhelm Erben (eds.): Practice and Theory of Automated Timetabling III (Third International Conference, PATAT2000, Konstanz, Germany, August 2000, Selected Papers). Springer Lecture Notes in Computer Science 2079, 2001
6. Edmund Burke and Patrick de Causmaecker (eds.): Practice and Theory of Automated Timetabling IV (Fourth International Conference, PATAT2002, Gent, Belgium, August 2002, Selected Papers). Springer Lecture Notes in Computer Science 2740, 2003
7. Edmund Burke and Michael Trick (eds.): Practice and Theory of Automated Timetabling V (Fifth International Conference, PATAT2004, Pittsburgh, PA, August 2004, Selected Papers). Springer Lecture Notes in Computer Science 3616, 2005
8. Tim B. Cooper and Jeffrey H. Kingston: The solution of real instances of the timetabling problem. *The Computer Journal* **36**, pages 645–653, 1993
9. Pascal Van Hentenryck: The OPL Optimization Programming Language. MIT Press, 1999
10. Jeffrey H. Kingston: A tiling algorithm for high school timetabling. In Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, PA, pages 233-249, August 2004
11. Jeffrey H. Kingston: The KTS high school timetabling web site (Version 1.3), October 2005. <http://www.it.usyd.edu.au/~jeff>
12. Jeffrey H. Kingston: The KTS high school timetabling system. Submitted to 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, Czech Republic, August 2006
13. W. J. van Hoeve: A hyper-arc consistency algorithm for the soft alldifferent constraint. Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004), Springer-Verlag Lecture Notes in Computer Science 3258, pages 679-689, 2004
14. D. de Werra: An introduction to timetabling. *European Journal of Operational Research* **19**, pages 151-162, 1985

An Approach for Automated Surgery Scheduling

Karl-Heinz Krempels and Andriy Panchenko*

RWTH Aachen University,
Computer Science Department - Informatik IV,
Ahornstr. 55, D-52074 Aachen, Germany
{krempe, panchenko}@cs.rwth-aachen.de

Abstract. The planning of surgical operations forms a substantial element of hospital management. It is characterized by high complexity, which is caused by the uncertainty between the capacity offered and the true demand. Also, as emergency cases occur the planning requirements change. A semi-automated dialog-based system is therefore preferred rather than either fully manual or fully automated systems. This is because of the inability of the later to recognize the changes in a high dynamic environment and to take the responsibility for decisions made. As it has to be possible to add new tasks in the planning process “on the fly” and to adequately plan new situations, we involve a human planner in the scheduling activity. The planner acts as a “sensor” to identify changes as they occur and integrates his knowledge as well as his decision-making competence into the planning process. The proposals for the schedules are however made with the help of the heuristics. In order for a schedule to be accepted by those involved, it should take account of the interests and preferences of all the human actors. Existing systems do not do this and therefore suffer from levels of non-acceptance of their resulting schedules. In this paper we discuss suitable heuristics for operating theatre scheduling, the limits to which preferences can be considered in the scheduling process, and the validation of the approach in an experimental set of hospital scenarios.

1 Introduction

Operating theatre scheduling deals with assignment of limited hospital resources (rooms, doctors, nurses, etc.) to jobs (patient treatments, surgery, etc.) over the time, in order to perform a set of tasks according to their needs and priorities, and to optimize usage of hospital resources [13]. The whole process is restricted by a whole series of constraints, limitations and preferences [2]. It is further characterized by a high level of complexity due to:

1. the uncertainty of the relationship between the capacity offered and the true demand,
2. the inability to predefine treatments’ workflow,

* This work was supported by the German Research Foundation in the research priority programme SPP 1083 – *Intelligent Agents in Real-World Business Applications*.

3. emergency cases, causing disturbance in existing schedules and the need to adapt as situation changes.

Typically, scheduling is currently done manually and involves specialized staff. [9] discuss an example of an optimization that can be achieved using process automation for nurses rostering¹. In the hospital considered, one person takes 3-5 full working days to produce the nurses' schedule for the period of one month. That is only to the level of assigning shifts. Similar results have been reported in other studies [4]. Existing industrial schedulers usually assume a predefined workflow. Furthermore, they do not take staff and patient preferences into account and so are not generally applicable to the hospital domain [2]. The current methods in this domain are therefore: no planning, pen and paper, non-intelligent tool-based. Fully automated systems are not accepted because of their inability to give proper consideration to preferences and the need to show clear responsibility for decision making.

Existing solutions consider only sub-problems such as: nurse rostering, coordination, or surgery planning. Most tools provide only GUI for manual scheduling, although some check the validity of completed schedules, and some of the more advanced generate draft schedules (however, preferences are not taken into account). Some examples of such systems are Medico//S, ORBIS², MEDICUS [14], CARE2X³.

This paper presents a problem, based on real-world requirements. In the following section we present our approach for semi-automatic, dialog- and preference-based scheduling in hospital scenarios. Then, after introducing the scheduling problem to be considered, the domain of discourse and the made assumptions in Section 2, we describe the problem decomposition, the discussion of the developed heuristics for each of the identified subproblems and provide a complexity analysis for the developed heuristics. In Section 3 the experimental context is described and an evaluation of the results presented. We finalize the discussion with conclusions based on the evaluation and look forward to further challenges.

2 Scheduling Heuristic

We divide the operating theatre scheduling problem into the four sub-problems: preference-based personnel assignment to shifts, preference-based building of teams in a shift, preference-based task (operation) assignment to teams and room assignment to tasks. For each of the sub-tasks we have developed a heuristic. Furthermore, we have implemented suitable problem-solving methods and evaluated the implementation by the means of simulation. The heuristics produce a proposal for the schedule. It is however up to the human planner whether to accept it in its entirety, accept parts of it, or to reject it completely. We propose a semi-automated dialog-based approach for the rostering.

¹ rostering and scheduling are used as synonyms here

² <http://www.sieda.com/>

³ <http://www.care2x.com/>

In this section we discuss the heuristics for operating theatre scheduling according to the division of sub-tasks as described above. We assume that the preferences of the actors involved that need to be taken into account are provided in a frame-based representation, and that their context will be valid in the context of the domain: shift, team, task, and room [10].

Based on the nature of the preference, we distinguish between static (coworker, treatment) and dynamic (shift) ones. Each preference has a value from the closed domain $\{willingly, undecided, unwillingly\}$. We define the preference domain range as

$$\mathcal{D} = \{w, u, \bar{w}\},$$

where w stands for “willingly”, u for “undecided” and \bar{w} for “unwillingly”. Each employee is able to specify a preference $x_i \in \mathcal{D}$ for another employee expressing his/hers willingness to work together with that employee. In the same way it is possible to specify preferences for shifts. If the preference is not specified it has the value u (undecided) by default.

There are basically two approaches for the staff rostering problem: *cyclic*⁴ and *non-cyclic* [1]. In the first one, several sets of schedules are generated that satisfy the demand requirements. Staff are then rotated from one set of schedules to another in consecutive planning horizons. So for example, the schedule may be repeated every week or with a two week interval. Although it is easy to implement, cyclic schedules impose inflexibilities, and there is therefore less acceptance of the resulting outcome. We therefore use a non-cyclic algorithms.

It is also important to note that there are two possible approaches for team continuity: first one based on the static team assignment, that means a team remains unchanged until the end of the duty; the second one allows the reassignment of members to other teams (build new ones) when no job is assigned to the current one, in order to achieve continuity of occupation. Static assignment is less flexible, but has much less computational complexity since there is no need to search for the common time gaps across all team members. Furthermore, in the second, there may be individual time gaps that will not be filled in the future with any tasks.

2.1 Preference-Based Adaptive Assignment of Personnel to Shift

Preference-based assigning of personnel to shift considers:

- hierarchical position (senior physician, assistant doctor, anaesthetist, nurse);
- contract work hours (treated as soft-constraints);
- shift preferences (that have dynamic character since preferred working time can vary from day to day);
- hard-constraints:
 - minimum/maximum number of personnel in the shift (day, shift and qualification dependent);

⁴ sometimes also called *rotational*

- maximum duty duration;
- minimum inter-duty pause;
- fulfilled wishes quota.

The heuristic selects personnel so as to avoid hard-constraint violations for each qualification in the following order of preference value:

$$\text{willingly} \succ \text{undecided} \succ \text{unwillingly}.$$

The best and simplest case is when the number of actors that work gladly in the shift is within the bounds of the hard-constraints and maximum working time is not exceeded. This means that they can all be selected for the duty and it can be continued with the next lower position in the hierarchy of the department. Unfortunately the hard-constraints often prevent this. So, for example, the number of available personnel that would work “willingly” or “undecided” in the shift may be smaller than the minimum staffing level required. In this case all of the actors with these preferences have to be selected and those with the lower preferences have to fill the gaps. It may also happen that only some of the actors with the preference “unwillingly” for the current shift have to be selected. So those that are not selected will be in a better position compared to the selected ones. Furthermore, in the case of selecting only some with the “willingly” preference, those that are not selected will find themselves in the worse condition compared to the others. The selection procedure therefore contributes to placing some actors in a worse condition, compared to the others. Such a decision has to be memorized in order to achieve a degree of fairness in the scheduling. That means that an additional measure has to be introduced in order to keep track about the number of actor’s wishes that have been, respectively have not been taken into account. The measure is mapped to the actor’s weight. This weight is then used to influence further selection decisions of the form “some from many” in order to achieve fairness of the scheduling and give due consideration to the wishes of individual actors. It should also be considered that medical personnel are usually contracted to work for a certain amount of hours per week. The selection process therefore also needs to take into account that physicians that have not yet completed their weekly contract hours should have higher priority in getting a shift than those that are already on overtime. The weight (importance) of the choice for an actor can be determined with the help of the following weight function:

$$\begin{aligned} \text{Weight}_i = & \text{RemainingWorkShifts} \cdot \text{PositionWeight} \cdot \alpha \\ & + \text{Weight}_{i-1} \cdot \text{stimulus} \cdot (1 - \alpha) \end{aligned} \quad (1)$$

where

- $0.5 \leq \alpha < 1$ coefficient that determines influence of the previous weight value. It is assumed, that the remaining working time has more influence on the weight function than whether wishes have been complied with, or rejected;

- $PositionWeight$ weight, determined by the position of the actor;
- $RemainingWorkShifts$ quota regular working shifts stated in the contract as represented by the soft-constraints;
- $Weight_{i-1}$ previous (old) weight value;
- $stimulus$ determines the weight adjustment manner:
 - > 1 in case of dissatisfying selection;
 - = 1 if no preference was specified;
 - < 1 in case of taking into account a wish.

$PositionWeight$ is used in the weight function since it is assumed that duties may require personnel with some qualification, independent of the position. However, the wishes of the physician with a higher position in the ward hierarchy are given a higher significance.

Defined as above, the stimulus effect decreases the weight function in the case of selecting an actor with the preference “willingly” or by not selecting one with the preference “unwillingly”. The value increases by selecting with “unwillingly” or by not selecting with “willingly”. The adjustment strategies of the weight with respect to stimulus can be seen in Table 1.

preference value	stimulus for selection	stimulus for rejection
“willingly”	< 1	> 1
“undecided”	= 1	= 1
“unwillingly”	> 1	< 1

Table 1. Weight Adjustment Strategies

The preference “undecided” is treated as neutral and does not influence the stimulus, but the weight function changes in case of selection because of the adjustment of remaining work shifts. At the beginning of the week the weight is initialized as:

$$Weight_0 = contractualWorkShifts \cdot PositionWeight \quad (2)$$

The weight is always adjusted each time the person is selected for the shift (since the $RemainingWorkShifts$ number changes) and also, for all those actors with a preference $\in \{willingly, unwillingly\}$ that were able to take the duty (i.e. caused no hard-constraints violation). For not selected actors only the stimulus will have an impact on the weight. If the $RemainingWorkShifts$ reaches the value zero, the weight function update decreases significantly if the actor is selected (since it is treated as an undesired overtime):

$$Weight_i = Weight_{i-1} \cdot stimulus \cdot (1 - \alpha) \quad (3)$$

In case of rejection, for those actors with a preference $\in \{willingly, unwillingly\}$ only the stimulus has an influence on the weight:

$$Weight_i = Weight_{i-1} \cdot stimulus \quad (4)$$

All relevant constraints and their characteristics are given in Table 2.

The minimum and maximum number of personnel required in the shift has a dynamic character since these values depend on the day (e.g. regular week day, weekend, etc.). Where more than minimum required personnel with a preference *willingly* are available, they are all selected up to maximum allowed number by the hard-constraint. In the case of preferences $\in \{undecided, unwillingly\}$ only the minimum number stated in the constraints are selected. The heuristic therefore satisfies the wishes of as many actors as possible. It is important to mention, that the personnel requirements have following restrictions:

$$\begin{aligned} \text{max required senior physician} &\leq \text{min required assistant doctor} \\ \text{max required assistant doctor} &\leq \text{min required nurse} \\ \text{max required assistant doctor} &\leq \text{min required anaesthetist} \end{aligned}$$

Where there is a need to make a selection decision “select some from many”, those with the higher value of the weight function are chosen. This contributes towards fairness, since higher values mean that there are more contractual working hours still unused in the current week, a higher position in the ward’s hierarchy or smaller number of preferences already taken into account.

In view of the linear simplicity of the heuristic we do not discuss the complexity analysis for shift assignment.

2.2 Preference-Based Building of Teams in a Shift

After the staff selection for the duties, the available personnel in the hospital department is determined for each shift. As tasks arrive, there is a need to group the shift personnel into teams in order to perform those tasks. Requirements for personnels’ qualifications, resources, specializations of the involved actors in each treatment have to be considered. In order to increase the acceptance of the planning system in the hospital, it is also important to take into account personal preferences of the involved actors. So, for example, the doctor “Z” may prefer to work better with nurse “Y” than with the nurse “X”.

Constraint	Type	Character
shift preference	preference	dynamic
max shifts per duty	hard-constraint	static
min inter duty pause	hard-constraint	static
min required personnel	hard-constraint	dynamic
max required personnel	hard-constraint	dynamic
contract work hours	soft-constraint	static

Table 2. Constraints for Shift Assignment

The problem can be formalized as follows: given S, D, E, N - sets consisting of senior physicians, assistant doctors, anaesthetists and nurses having duty in a shift, so that:

$$|S| = s, |D| = d, |E| = e, |N| = n \quad \text{with } s < d < e < n.$$

This means that there are always more nurses in the shift than anaesthetists and more anaesthetists than assistant doctors, and more assistant doctors than senior physicians. Each employee can specify a preference with respect to another, expressing his willingness to work with them:

$$\forall a_i, a_j \in S \cup D \cup E \cup N, \quad i \neq j \quad \exists \mathcal{P}_{a_j}(a_i) = x_{ij}, \\ x_{ij} \in \mathcal{D}, \quad i, j \in (1, s + d + e + n).$$

Every employee is also characterized by the weight, that depends first of all on the hierarchical position of the person:

$$\forall a \in S \cup D \cup E \cup N \quad \exists g(a) \geq 0.$$

The utility of team i , consisting of \tilde{n} actors is defined as

$$U_i = \sum_{k=1}^{\tilde{n}} q_{a_1, \dots, a_{k-1}, a_{k+1}, \dots, a_{\tilde{n}}}(a_k) \cdot g(a_k) \quad (5)$$

where

$$q_{a_j}(a_i)_{i \neq j} = \begin{cases} 1 & \text{if } \mathcal{P}_{a_j}(a_i) = w, \\ 0 & \text{if } \mathcal{P}_{a_j}(a_i) = u, \\ -1 & \text{if } \mathcal{P}_{a_j}(a_i) = \bar{w} \end{cases} \quad (6)$$

is the value of the coworker preference of actor a_i regarding the colleague a_j . Correspondingly, for a set of n coworkers a_1, \dots, a_n it is defined as

$$q_{a_1, \dots, a_{k-1}, a_{k+1}, \dots, a_{\tilde{n}}}(a_k) = q_{a_1}(a_k) + \dots + q_{a_{k-1}}(a_k) + q_{a_{k+1}}(a_k) + \dots + q_{a_{\tilde{n}}}(a_k). \quad (7)$$

The total utility of h teams in a shift, each with its own size of \tilde{n}_i is defined as

$$U_{total} = \sum_i^h U_i = \sum_{i=1}^h \sum_{k=1}^{\tilde{n}_i} q_{\{a_1, \dots, a_{k-1}, a_{k+1}, \dots, a_{\tilde{n}}\}_i}(a_{ki}) \cdot g(a_{ki}). \quad (8)$$

The goal is now to build teams in such a way, that the total utility function (satisfaction of coworkers to work together) is maximized.

Optimization Criteria. It is assumed that there exist two configuration patterns for operation teams: those with four actors (senior physician, assistant doctor, anaesthetist and nurse) and, for less complicated operations, consisting of three actors (assistant doctor, anaesthetist and nurse). The goal is to find teams, consisting of staff members from a shift, and maximizing the total utility function.

So, for example, in case of preference and weight specifications as captured in Table 3 the utility of the three-member teams is calculated as follows:

$$\begin{aligned} \mathcal{U}(< A, B, C >) &= q_{BC}(A) \cdot g(A) + q_{AC}(B) \cdot g(B) + q_{AB}(C) \cdot g(C) \\ &= (0+1) \cdot 80 + (-1+1) \cdot 60 + (-1+0) \cdot 40 = 40. \end{aligned} \quad (9)$$

The total utility is then the sum of weighted adjacent satisfaction of all the team members. So, the advanced teams are built up for all available senior physicians. For the rest $t = 1, \dots, d - s$, assistants (where $s = |S|$, $d = |D|$) smaller teams are built in a similar way, in order to facilitate treatment of another (lower) complexity class. The maximum sum reflects the highest total utility of all the teams involved and would be the optimal solution for the problem.

Complexity Analysis Discussion. To achieve the maximum utility value, it is necessary to compare the sums of all the possible team configurations in the shift. The total number of possibilities for a set of s teams of four members in the shift is

$$\sum_{i=0}^{s-1} (s-i)(d-i)(e-i)(n-i), \quad (10)$$

which requires a computational complexity of $O(s \cdot d \cdot e \cdot n)$, or $O(n^4)$.

It is important also to consider what happens when the team size is not fixed. The complexity results are completely different. The problem can be represented as a graph where each vertex represents a team configuration and an edge between two vertices exists in cases where they have at least one common actor. The goal is to find maximum independent set in the graph. An *independent set* in a graph $G = (V, E)$ is a subset $I \subseteq V$, such that $\forall x, y \in I$, $\{x, y\} \notin E$. The independent set problem consists of finding a maximum (largest) independent set in a graph. It is well-known to be NP-complete [7]. In natural integer programming formulation the finding of the optimal solution with team weights w_j

Employee x	Preference to A $P_A(x)$	Preference to B $P_B(x)$	Preference to C $P_C(x)$	Weight $g(x)$
A	-	u	w	80
B	\bar{w}	-	w	60
C	\bar{w}	u	-	40

Table 3. Example for Team Utility Calculation

for $j \in V$ (V is a set of all possible team combinations in a shift), $|V| = s \cdot d \cdot e \cdot n$, is

$$\begin{aligned} & \text{Maximize } \sum_{j=1}^n w_j x_j \\ & \text{subject to } x_i + x_j \leq 1 \text{ (for every edge } (i, j) \text{ in the graph)} \\ & \quad 0 \leq x_j \leq 1 \quad (j = 1, \dots, s \cdot d \cdot e \cdot n) \\ & \quad x_j \text{ integer} \quad (j = 1, \dots, s \cdot d \cdot e \cdot n) \end{aligned}$$

Hochbaum([7]) gives a summarization of all the approximation algorithms for the weighted independent set problem known to date. A guaranteed good approximation is only possible if the graph has some special features such as being planar (in this case it is possible to give an approximation guarantee of $\frac{1}{2}$). In the general case it is of the $\lceil \frac{1}{\lceil \frac{\Delta+1}{3} \rceil} \rceil$, where Δ is the maximum degree of a vertex in the graph. In the case of a teams graph where teams consist only of four actors, the degree of each vertex is

$$s \cdot d \cdot e \cdot n - (s-1)(d-1)(e-1)(n-1) - 1.$$

This is nothing other than a total amount of all possible combinations of teams, having substituted those combinations, not having at least one common actor with one selected team. If the hospital department consists of 25 members in each position, the vertex degree Δ has the value 58848, considering only teams of four actors. In this case the approximation degree of about 0.00005 or $5 \cdot 10^{-5}$ can be guaranteed with the complexity $O(m\Delta)$, which does not seem to inspire much (m is the number of edges). Guo et al. ([6]) has experimentally compared and found that using the simulated annealing heuristic to solve the set packing problem (which is polynomially equivalent to the independent set problem [7]) outperforms previous approximation methods, and based on that heuristic ILOG CPLEX obtains results within smaller timescales.

Heuristic. Now that the problem formally defined a feasible solution can be introduced. It is guaranteed to be pareto-optimal (it is not possible to increase the utility of one team, without decreasing the utility of another one), but not the optimal in the general case, since long computations are required to handle the very large numbers of different shift team combinations that need to be considered (see Formula 10). An obvious solution, in order to achieve pareto-optimality, is to calculate for each shift the utility for each possible team combination, sort it, and take those with the highest utility value. However, doing this dynamically each time for each shift requires many computations and an extended waiting time before seeing a proposed schedule. It can be improved by calculating the utility for all possible team combinations in the department only once, since the coworker preference bears a static character and is not changed often. The complexity of such a computation is $O(s \cdot d \cdot e \cdot n)$ or $O(n^4)$ since n is the largest of these four numbers. When any actor changes their preferences, the recalculations required are of $O(n^3)$. Having constructed the array of utilities it is sorted into order so that teams with a higher utility are preferred over those with a lower one. The *heap sort* is selected to facilitate the process. It is the slowest of

the $O(n \lg n)$ sorting algorithms, but unlike the *merge* and *quick sorts* it does not require massive recursion or multiple arrays to work. This makes it the most attractive option for very large data sets of millions of items.

Every time teams for a shift are built, it is now sufficient to process the sorted array in descending order of team utilities until all the senior physicians (in case of building advanced teams) or assistant doctors (in case of building teams for less complex treatments) are assigned. The heuristic checks if all of the team members considered have a duty in the shift and are not yet members of a team. If this is the case, the actors are assigned to work in one team, since no other combination of available actors can produce a higher current team utility value (remembering that the array is sorted). This yields a pareto-optimal solution. In the worst case there is a need to go through all of the elements in the array.

Furthermore, the solution obtained can be improved with the help of a modified simulated annealing heuristic. It can often be the case that it is not optimal in the sense, that another combination of teams may exist, so that the total shift utility is higher, than that achieved with the pareto-optimal algorithm.

The maximum value of the team utility function will always be less than the product of the greatest team weight and the number of teams that have to be build ($|D| = d$). However, a better approximation can be reached taking the sum of the first greatest d elements of the sorted teams weight array. Even this yields a value that is not less than the optimal solution (since these d teams are often not disjoint). It serves as a probabilistic termination criteria if the obtained solution is close enough to the calculated bound. The heuristic itself starts on the i -th iteration with the $2i$ -th element of the descending sorted array of the actors present in the shift, and moves at first upwards, then downwards from the $2i$ -th element, checks whether the current team candidate members are not yet assigned for a team, and in the case of positive outcome, selects them. At the same time the utility function is calculated. If the current utility of the sum of d teams is greater than the known maximum, then the current teams assignment is chosen as a candidate for the shift assignment and it is continued with the next iteration. If no improvement is achieved, the temperature value is decreased.

2.3 Preference-Based Task Assignment to Teams

After the preference-based resource allocation (planning phase, building teams) it is now possible to proceed with scheduling. At first the problem analysis will be given, afterwards the heuristic will be discussed.

Problem Analysis. The problem of scheduling we are faced with can be formulated as follows: given a set of v teams, that have duty in the shift x as well as the set of q tasks, that are planned for that shift. Each task has an approximate duration (since it is often not possible to forecast how long an operation will take due to non-deterministic workflow), contributing to the uncertainty degree of the schedules generated. It is also characterized by its priority in order to distinguish between emergency and regular cases. The main component of each

task is a patient, that is characterized by insurance (private or governmental), preferred treatment time as well as a doctor preference. Each team is restricted within the shift by the maximum working time allowed without a break and a minimum break duration as stated in hard-constraints. It is also characterized by the time it finishes its last task as currently scheduled. A graphic representation of the task scheduling problem is captured in Figure 1. The goal is to assign those q tasks to v teams in such a way that no hard-constraint is violated and the objective function gets the highest value.

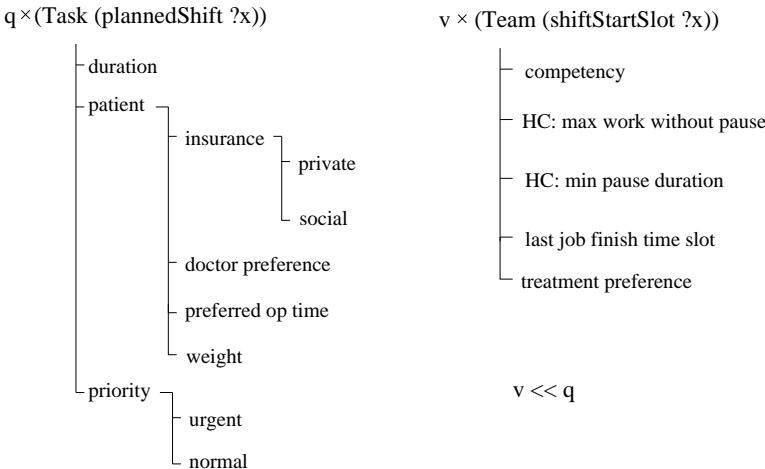


Fig. 1. Task Assignment Problem

The scheduling problem is defined in the terms of the Graham's notation [12]:

Constraint / Preference	Type	Character
max slots without pause	hard-constraint	static
min pause duration	hard-constraint	static
patient doctor preference	preference	static
patient preferred treatment time	preference	dynamic
doctor treatment preference	preference	static
team competency	hard-constraint	static
task priority	hard-constraint	dynamic
patient insurance	soft-constraint	static
shift end time	hard-constraint	static

Table 4. Constraints for Tasks Scheduling

$$Rm \mid r_j, M_j, brkdwn \mid \theta_1 \sum w_j T_j + \theta_2 \sum \hat{w}_j Z_j$$

The first element of the triple - machine environment - refers in the case of the operation theatre scheduling to the team environment. There is no exact environment notation to express the situation faced with in the team scheduling. That is why the most suitable environment is taken with the assumption noted below. Rm stands for unrelated machines in parallel. Usually it is used if the machines have varying speeds and that speed of depends on the job being executed. However, the speed of performing an operation is not the selection criterion for the teams, and that is why it does not play any role here. Job j requires a single operation and may be processed on any of the machines belonging subset M_j , however, different machines (teams) yield different values regarding optimality criterion due to the needs of taking into account team and patient preferences. The processing restrictions include release dates (r_j), machine eligibility restrictions (M_j), and breakdowns ($brkdwn$). The first means that the job j can not start before its release date r_j . So, for example, patients, planned to be operated on some specific date, can not be scheduled before that date. Machine eligibility restrictions mean that not all m machines (teams) are capable of processing job j (some specific kind of operation). The set M_j denotes the set of machines that could process the job j . Breakdowns imply that machines (teams) are not continuously available (due to e.g. pauses, times for operation room preparation). As the objective to be minimized is the total weighted tardiness ($\sum w_j T_j$) plus the total weighted team (treatment type) and patient (senior physician) dissatisfaction ($\sum \hat{w}_j Z_j$) chosen, since it conforms the requirements of the operation completion times and team's and patient's wishes in the hospital. The composite objective includes also weights θ_1 and θ_2 for each of the two sub-objectives. So the goal is to minimize the due date violations, where weight w_j may be used to specify the priority of the job e.g. urgent, normal, etc., as well as \hat{w}_j to specify the importance of taking into account wishes of corresponding actors.

After having formally defined the problem, it is now possible to define its complexity. It is made with the help of reduction to the *3-partition problem* [5]. The 3-partition problem is well known to be strongly NP-hard [12]. Strongness means that the problem cannot be optimally solved even by an algorithm with a pseudo polynomial complexity. The proof idea can easily be derived from Pinedo [12].

Since even a simplified scheduling problem with only one machine (team) without any preferences and room considerations is NP-hard, the more general problem with many teams with preferences for treatments, coworkers and time consideration is therefore not less difficult.

Heuristic for Preference-Based Scheduling of Tasks. This heuristic selects tasks that are planned for the shift considered and schedules first the urgent ones, and then the regular ones. Scheduling the urgent tasks tasks before the

regular ones matches the normal prioritization in a hospital department. The schedule is developed with the help of *dispatching rules*.

The dispatching rule determines which task should be scheduled as next. Every time a team becomes free and the time since the last index update exceeds some Δ , the ranking index is recomputed for each remaining job. Their list is then sorted and the jobs with the highest ranking index are processed first. This ranking index is a function of the time t , at which a team with the earliest last job finish time becomes free, as well as the patient weight w_j , task processing time p_j , task urgency u_j , and the due date d_j of the remaining jobs. The index is based on [12] and defined as

$$I_j(t) = \frac{w_j \cdot i_j}{p_j} \cdot e^{-\frac{d_j - p_j - t}{K\bar{p}}} \cdot e^{u_j} \quad (11)$$

where

w_j is the patient's weight, that is assigned to the task j ;

i_j is the insurance type of the patient from the job j ;

p_j is the task j processing time (duration);

d_j is the due date - time, until the job has to be done;

t is the time, machine can begin processing at;

K is a scaling parameter;

\bar{p} is the average of processing times of the remaining jobs;

u_j is the urgency of task j .

The first part of the index ($\frac{w_j \cdot i_j}{p_j}$) determines the price that a patient is paying for one slot of time for his task. The second two parts of it have exponential character and represent the task urgency and the slack influence on the index. K is the *scaling parameter* that can be found empirically and determines the influence of the first exponent on the index function. The smaller K is, the higher is the influence of the first exponent on the whole index. Sometimes K is also called *lookahead parameter*. The *slack* of the job j is the time left before the latest time point the job should be started, in order to do not exceed the due date. It is less than zero if the job can no longer be finished before the due date. This means that the smaller the value, the greater the due date violation. If the slack is positive, the first exponent decreases the value of the index function. Task urgency u_j can be e.g. equal zero for regular tasks, have value greater than zero for urgent ones, having the greater value the more urgent concrete task is.

After the weight calculation for tasks planned for the considered shift is completed, the task list is sorted in the descending order of the weight. In this order the tasks are further processed so that those with a higher weight get scheduled first. The task under consideration is tested in order to determine whether the patient has specified the preferences for the operation chief (senior physician in case of teams having four actors, assistant doctor in case of smaller teams consisting of three actors). If it is the case, we check whether the physicians as specified by the patient preference are present on the shift. Those present are then further categorized into sets with preference "willingly", respectively "unwillingly" (all other belong to "undecided"). Further, each of the sets is

sorted in the ascending order of the teams' last task finish time and stored to an ordered list. This contributes to the attempt to schedule the currently less occupied teams within a preference value first. Having finished sorting, the lists \mathcal{L}_w , \mathcal{L}_u , and $\mathcal{L}_{\tilde{w}}$ are concatenated to the list \mathcal{L} in the following order of preference value:

$$\mathcal{L} = \text{CON}\{\mathcal{L}_w, \mathcal{L}_u, \mathcal{L}_{\tilde{w}}\}.$$

If the current task to be scheduled is an urgent one, the earliest possible starting time is calculated without regard for the specified preference. Preferred teams are only then considered if their starting time deviated from it by not more than a small value ε since these tasks have to be executed as soon as possible:

$$\text{currentTask}_{ts} - \text{earliestPossible}_{ts} < \varepsilon, \quad \varepsilon > 0. \quad (12)$$

For regular tasks the physician preference has more influence than with the urgent ones, since it is more important to take this preference into account, even if the task will be scheduled later but still within the shift.

Next we process the ordered team list and select a team for the job. An attempt is made to schedule the team for the next possible time. If this fails because the team needs to take a break, the break is scheduled and we retry. If it fails for the second time, the next team is considered. If the list is processed to the end and if no team was chosen - the dialog-based scheduling mechanism has to take over.

In case of success, the selected team is assigned to the appropriate position in the list of shift teams, to keep it in ascending order of the finishing time of the last task. Scheduling proceeds with the next job.

We introduce a function of the following arguments to handle the treatment preferences of the team:

- doctor preference of the patient;
- last task finish time slot of the team;
- treatment preference of the team.

During the task scheduling, the function value is calculated for each team. The teams are considered in order of the returned function value, rather than being divided into three groups with patient preference values from \mathcal{D} and sorting regarding the last task finish time within the group.

Discussion. An important feature of the task scheduling mechanism proposed is that it allows rescheduling on demand. The dynamic character of job arrivals in a hospital environment causes disturbances to pre-existing schedules, making reactive scheduling necessary. The plan is being adapted to new situations as the changes occur. However, the emphasis is to keep as much as possible of the existing plan untouched. Rescheduling also takes place when an urgent task is added to the system. In this case all tasks, that have already begun are kept unchanged. In contrast, all un-started ones have their status changed to *unscheduled* and are re-proceeded by the scheduler. Following, further possible improvements can be considered for the job assignment:

- if a hard-constraint is violated due to the maximum work time without a pause, it is possible to search for a shorter task to fill the gap instead of recommending a pause right away. This helps to minimize unneeded pauses;
- tasks, to which patients did not specified doctor preferences, can be postponed initially. Instead, the number of tasks with explicit preferences for each team should be computed. Scheduling those tasks without preferences to those teams that have less patients in queue would bring better overall satisfiability of the proposed schedules. However, not specifying the preference should not cause longer waiting times before processing;
- *team rotation* is an important feature for long and complicated operations. In real life some treatments last longer than one team can complete either due to the ending of the shift or due to exceeding the maximum allowed hours without a break. In this case a team “handover” must be performed during the task execution.

The proposed heuristic makes proposals for the task scheduling. However, it is up to the human scheduler whether to accept or to reject it. The goal is to minimize the manual rescheduling so that the proposed schedule is changed, if at all, only slightly. Tasks, that were not able to be scheduled automatically must be proceeded by the human anyway.

2.4 Room Assignment

The last stage in operating theatre scheduling covers the assignment of operating rooms. This approach is subject to the condition that there are enough rooms available in that their number at least equals the number of teams in the shift and furthermore, the room, assigned to the team, has all the equipment and facilities that the team may need for executing its tasks. In another case, if the teams are kept only for operations and no rooms are available, personnel resources are wasted. However, teams often pause between consecutive operations. Each of these time gaps can be too small to reserve the room for another task, but some optimization of these time windows is possible, such as maximizing the available gaps between operations already scheduled. It is assumed that the rooms considered for the optimization are all of the same type, so that it does not matter which room is chosen from the proposed set, and it is not vital for the personnel. Of course the hospital management goal is the full utilization of the resources, but at the same time a reserve capacity is required in order to handle emergencies and to deal with the device/room breakdowns. It is assumed that all the team and time assignments are already made. The algorithm chosen to facilitate the process comes from Kandler [8], who proposed it for scheduling in a virtual enterprise. The important feature for the room assignment is that the jobs are not shifted in time but retain their scheduled time slots unchanged.

3 Analysis and Comparison

In this section the proposed heuristics are evaluated in practice. The most interesting criteria are the time needed to produce a valid schedule proposal as

well as the quotient of regarded/disregarded wishes of the personnel, and its comparison to existing schedules produced by human actors (here the random assignment is used because this reflects reasonably well the reality in todays world where preferences are not considered). All evaluations were performed on an Intel Pentium 4 CPU 2.40 GHz with 512 MB RAM running SUSE Linux with default kernel version 2.4.21. The experimental setup was built using the multi-agent system JADE 3.2⁵ and the rule-based expert system JESS 6.1⁶.

3.1 Shift Assignment Evaluation

The measures for shift assignment are made for 20 different values for the personnel size in the department (from 10 to 200, with the step 10), each with four different required staff quotients (specified as a hard-constraint) in the shift. The quotient is 20%, 30%, 40%, 50% of the overall number of personnel.

The number of actors for each position is based on the figures given in Table 5. For example, in case of 200 actors, 20 are senior physicians, 40 are assistant doctors, 60 anaesthetists, and 80 nurses.

quote	qualification
10%	senior physician
20%	assistant doctor
30%	anaesthetist
40%	nurse

Table 5. Personnel Quote of Specified Position

Shift preferences are generated for all actors and shifts as follows: with the probability $\frac{1}{3}$ the preference for the considered shift is generated by an actor. If it has been generated, it is assigned a random value from the domain \mathcal{D} (each with the equal probability of $\frac{1}{3}$).

The number of required personnel usually varies due to different days and shifts. Thus, we distinguish between a shift $\{\text{early}, \text{late}, \text{night}\}$ as well as the day of the week (regular working day, weekend or holiday). Furthermore, for each of the days and qualifications, hard-constraints regarding the required staff number are usually set flexibly, e.g. between 8 and 10. That is, there must be at least 8, at most 10 actors with some qualification in the shift. Following the heuristic, the maximum number is only taken if all wish to work in the shift, no other hard-constraint is violated and the contractual work hours are not yet exceeded (soft-constraint). Otherwise, the lower number of personnel is selected. For the simulation, however, the upper and lower bound are kept identical.

⁵ Java Agent Development Framework, <http://jade.tilabs.it/>

⁶ Java Expert System Shell, <http://herzberg.ca.sandia.gov/jess/>

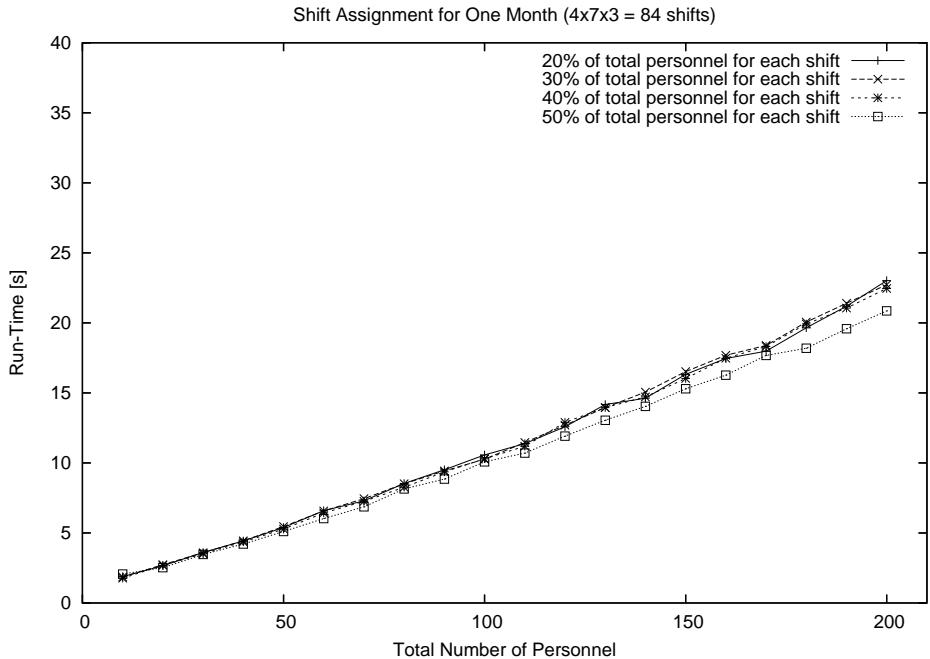


Fig. 2. Shift Assignment Run-Time

For each qualification the maximum duty duration as well as the minimum inter-duty pause hard-constraints were set to be equal to 16 hours. That is, it is not allowed to work more than 16 hours consecutively, and between 2 shifts must not be less than 16 hours. The soft-constraint that represents the contractual work hours is set to 40 hours per week for each employee.

Shift assignment is performed for one month (4 weeks, each with 7 days, each with 3 shifts that results in 84 shifts). The total number of staff in the hospital is chosen as a parameter. Another parameter is the number of required personnel per shift (four different configurations). The time required to produce such a schedule grows linearly with the number of personnel, and the assignment for one month takes less than a half of minute of computing time for 200 actors as captured in Figure 2.

The important criterion for acceptance of the proposed plans by human actors is the degree to which their individual preferences are taken into account. The presented simulations are made for the personnel of 200 actors selecting 10%, 30%, 50%, and 70% of total number for each shift for the period of one month (84 shifts). Figure 3 shows the percentage of preferences met in each shift. For a shift considered, the total number of preferences is calculated as the number of actors that have specified a preference with a value from $\{willingly, unwillingly\}$ for the shift. It is important, that only those actors are playing a

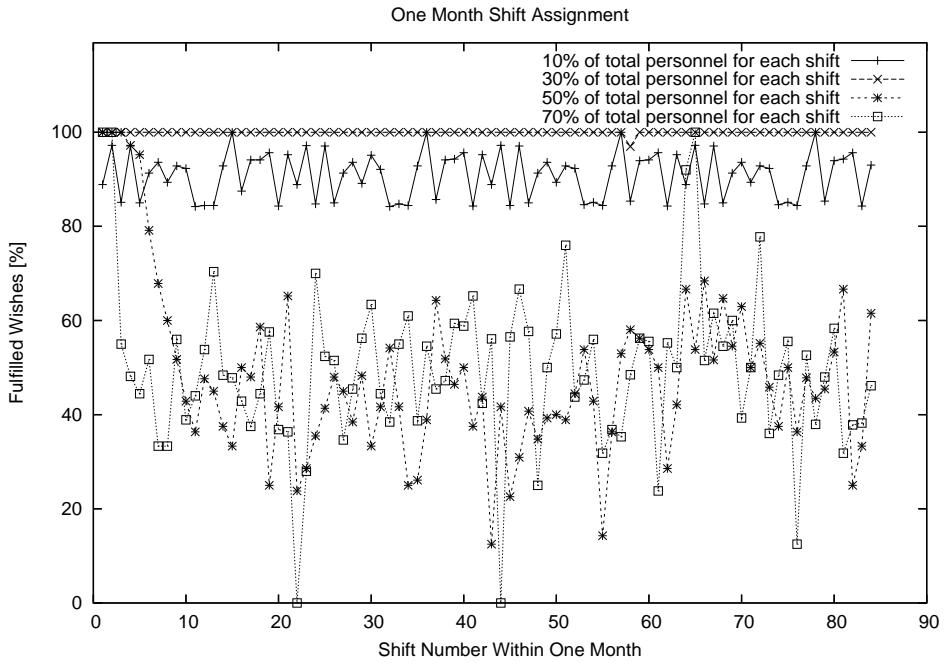


Fig. 3. Shift Assignment Fulfilled Wishes Quota

role here that are allowed to work in the shift (no hard-constraint violation). A preference is treated as respected in case of

- selecting an actor with preference value *willingly*;
- not selecting an actor with preference value *unwillingly*.

For 30% of total personnel in each shift all wishes are almost always fulfilled. In case of selecting 10% usually none of the unwillingly actors is scheduled for work, but not all who would like to work are selected, causing a slightly lower quotient of the preferences met compared to the previous case. Choosing the selection rate to 50% and 70% respectively of total personnel in each shift causes all the willing actors to be selected as well as some undecided and, eventually some unwilling too. Furthermore, selecting these numbers of staff causes violations of hard-constraints in some shifts due to the days off need. This leads to understaffing in these cases. Beyond that, often those who would like to work in the shift can't be even considered due to the above mentioned need for time off. Therefore actors with other preference values are selected. This contributes to the variation of the percentage of the preferences met. The amplitude of this variation is greater when selecting 70% of the entire staff. Nevertheless, the overall quote of preferences met is high and the proposed plans are likely to be accepted by the personnel, as compare to those manual schedules produced by human actors that usually disregard most wishes.

3.2 Team Assignment Evaluation

We need to initialize the team utilities before we can begin with team building. The initialization is done for the teams consisting of four and three actors. Due to the fact that the number of team combinations grows exponentially with the number of personnel, this is the most time consuming procedure in the algorithms. However, in a real life scenario the initialization must be performed only once. The changes are only needed if one decides to change one of his coworker preferences. In this case only one dimension of the utility array has to be recalculated. In Figure 4 the initialization and sorting times for teams consisting of four actors are captured.

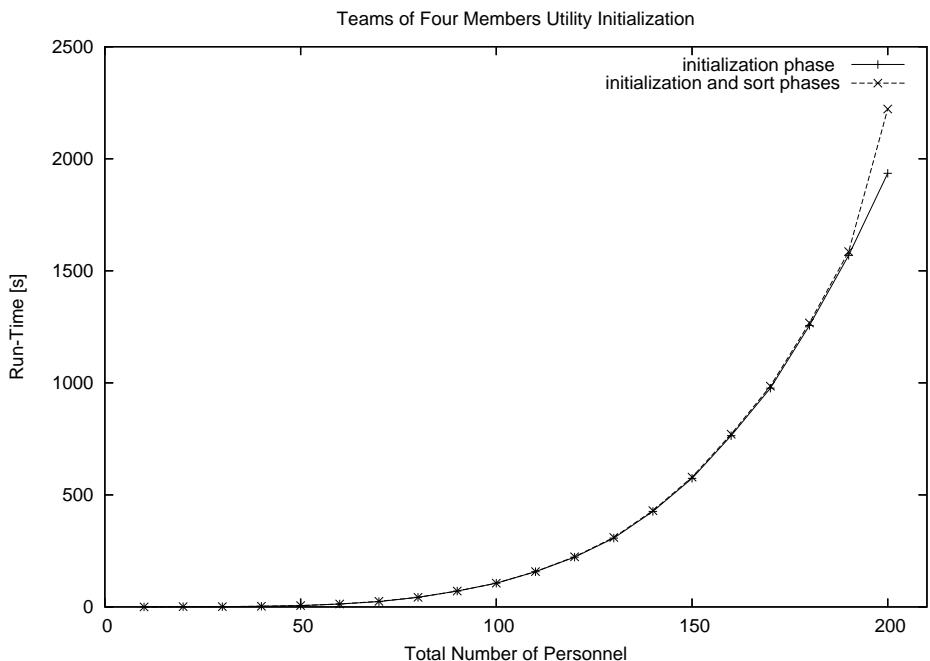


Fig. 4. Teams of Four Members Utility Initialization Run-Time

As can be seen from the figure, most time is spent on the initialization of the the team utilities and not on sorting. It is due to the need to query the knowledge base for each of the possible team combinations for the preference values of each actor regarding his coworkers. In case of three actors these are $3 \cdot 2 = 6$ queries and $4 \cdot 3 = 12$ queries for teams of four actors. For teams of four members, however, a significant deviation of the initialization time from initialization and sorting time is visible if the personnel size is 200 (the number of different team combinations is then $20 \cdot 40 \cdot 60 \cdot 80 = 3.840.000$). Repeated

simulations provided the same outcome. Detailed analysis has shown that with this number of combinations swapping occurred.

Simulation was performed in Java, instructing the Java Virtual Machine to initially assign 72MB allowing up to 1GB. The total number of staff was chosen analogously to the case of shift assignment. Coworker preferences are generated as follows: about 50% of the actors are selected randomly to specify a coworker preference. Each selected actor specifies preferences to roughly 20% of his random coworkers. These preferences have values either *willingly* or *unwillingly* each with the probability of $\frac{1}{2}$. For team utilities calculation we used the weight values from Table 6. For a ward with 200 actors the initialization and sorting time of the

weight	qualification
100	senior physician
80	assistant doctor
60	anaesthetist
40	nurse

Table 6. Team Assignment Personnel Weight

team utility array for teams of 4 actors takes less than 40 minutes, for teams of 3 actors less than one and a half minute.

In contrast to the team initialization and sorting that has to be performed only once, team building (assignment) is executed for each shift. In Figure 5 the time needed to build teams of four actors for one week is captured. The worst case occurs if there is a need to go down to the last element in the list of teams, sorted in descending order of team utilities. The other measures are made for the actual time needed to assign teams, selecting 20%, 30%, 40%, and 50% of the total personnel in each shift. Interesting questions that arise analyzing the graphs are why the actual measures are well below the worst case, and why it takes longer to assign less (selecting 20% of total) than more teams (50%). The importance of a senior physician (represented by the weight) is clearly greater than that of the other team members, leading to positioning team configurations with his favored coworkers in the beginning of the sorted team array. The more personnel is present in the shift, the higher the chances that the people he prefers and that prefer him are also in the shift. The search ends sooner due to the availability of teams that are located in the beginning of the array. Even increasing the number of teams (the number of senior physicians increases as well) does not contribute to longer run-times since in the beginning senior physicians are iterated in the array due to their great influence.

Next, the question arises how much improvement the proposed team assignment heuristic brings. It is not very obvious what kind of data should be taken as a reference to compare with. On the one hand it is important to satisfy as many coworker wishes as possible. On the other hand, the hierarchy has to be taken into account, because the proposed team building heuristic may not be

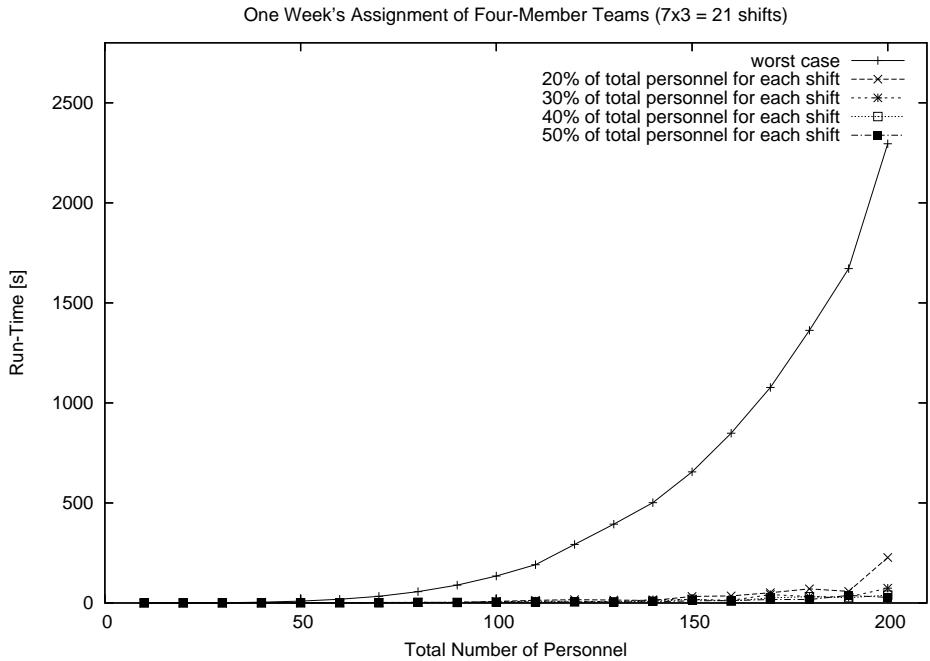


Fig. 5. Assignment Time of Four-Member Teams (With Worst Case)

accepted in the ward if the wishes of e.g. two nurses have always more influence on the decision than that of one senior physician. For demonstration purposes, however, in order to be able to compare the number of preferences taken into account in the teams, the weight of each preference, independently of the position and qualification of an actor, is set to be equal one:

$$\forall a \in S \cup D \cup E \cup N, \quad g(a) = 1.$$

As a reference for comparison, randomly built teams are chosen. Measures are made for one week (21 shifts) and show the sum of team utilities within the shift for the heuristic and random team building. The average values as well as the number of teams in each shift are also captured in the diagram. Figure 6 shows the simulation results selecting 40% of total personnel in each shift for building teams of four actors, with the size of the ward equal to 200 actors.

3.3 Task Assignment Evaluation

Last, but not least, job assignment is performed and evaluated for different numbers of staff in the shift, for the period of one week. Tasks are generated in such a way, that there are more jobs than the teams can perform in each of the

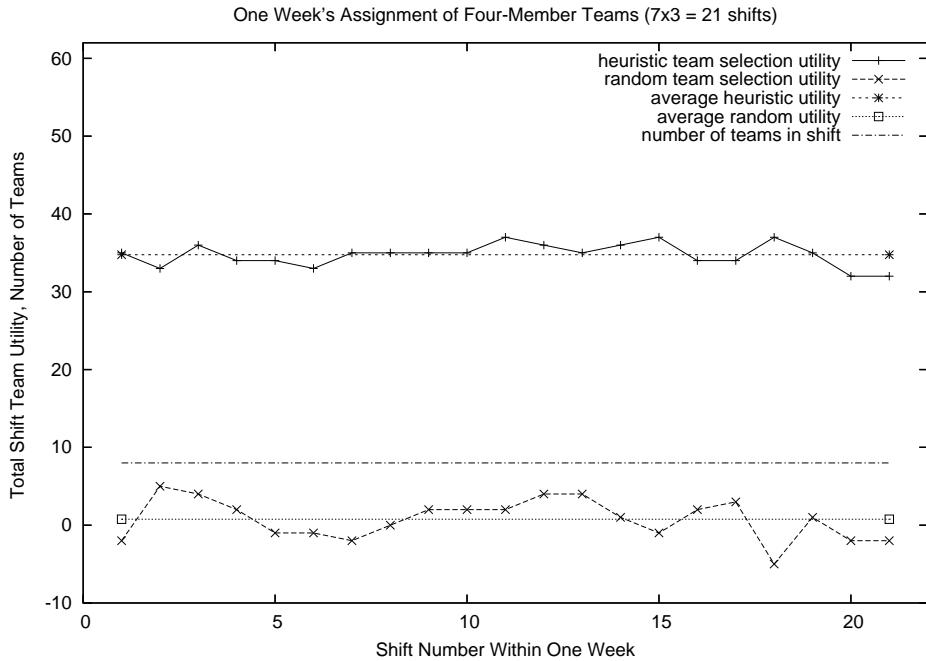


Fig. 6. Total Utility of Four-Member Teams (40% of staff for each shift)
Comparison of Heuristic and Random Assignment

shifts. In case of dealing with teams consisting of four actors, there are 10% of senior physicians in the department. Each shift has 32 slots (15 minutes one slot, eight hours per shift), each operation or treatment lasts at least one time slot (however, the duration of the operation is randomly generated from one to five time slots long). The product of the senior physician number and the number of slots within the shift gives the number of patients in the shift, for each of whom a random task is generated. With the probability 0.2 the task is urgent. Measures for the job assignment are captured in Figure 7 and performed for different numbers of staff and shift selection quotient. Each team is restricted by hard-constraints to operate a maximum of 4 slots consecutively (one hour) and has to take at least 4 slots off afterwards. The more teams in the shift, the longer the task assignment lasts. The higher number of possibilities for the teams, task can be proceeded at, causes the increase in required calculations. For 200 personnel in the ward the job assignment for one week (21 shifts) takes less than 6 minutes with the proposed heuristic.

4 Discussion

All the techniques described are heuristics that do not guarantee to find an optimal result. Instead, they aim to find a reasonably good solution in a relatively

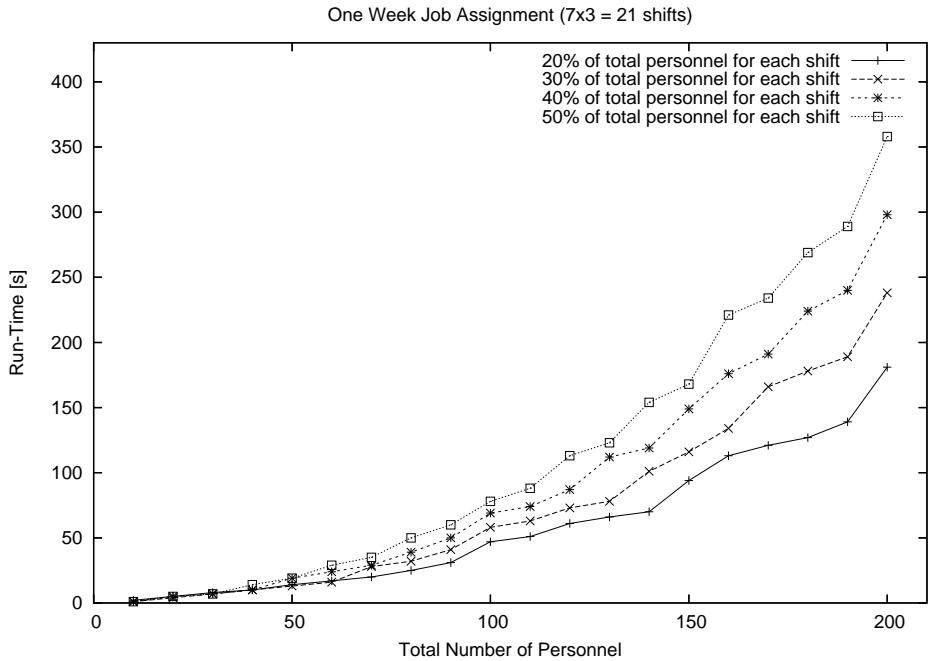


Fig. 7. Job Assignment Time

short time. The presented algorithms facilitate shift assignment (choosing personnel to work in a shift), team assignment, task assignment as well as room assignment. An important attribute of the heuristics is the consideration of the preferences of the involved actors as well as fairness due to the introduced weight functions.

Different heuristics could be developed in order to facilitate the scheduling process. For example, another possible approach for team building could be to use the features of an expert system by defining queries in order to find tuples of team member that match with a given satisfiability value and are on duty in the shift. The disadvantage of this approach is the impossibility to directly search (match) for the team with the highest utility, since only preference values have an influence on such a kind of pattern, not the weight an actor places on the concrete preference. The number of queries required to fetch all possible team preference combinations would be n^m , where n is the number of team members, and m is the number of possible preference values.

5 Conclusion

Staff timetables in medical departments are subject to lots of constraints, restrictions, and preferences [3]. Scheduling of hospital personnel is particularly

challenging because of different staffing needs on different days and shifts, uncertainty between the offered capacity and the true demand. Furthermore, it is impossible to predefine a treatment's workflow. As emergency cases occur (causing disturbance in existing schedules), there is a need of adaptation to situation changes. Due to the complexity and uncertainty the applicability of traditional (operations research and AI) methods from industrial scheduling to the operation theatres scheduling is problematic [11, 2, 10]. Usually a specialized person is in charge of this task (medical director). Yet, this often does not takes into account preferences of individual actors.

We have split the original problem into sub-problems and provided a preference-based adaptive heuristics for each of them. The system makes a schedule proposal and it is up to the responsible human actor either to accept, accept parts of the proposition, or to reject the schedule. In this paper we show, that the required time for shift, team, job and room assignment is within acceptable ranges for a real-world ward size [10]. The comparison of the heuristic approach to the random assignment was given. This allows to conclude that the proposed algorithms bring a substantial improvement regarding the number of fulfilled wishes of the actors, while planning and scheduling, and helps to save expensive human resources that are currently used in hospitals for the manual scheduling process. However, the preferences of the involved actors were randomly generated. It can often be the case that e.g. some actors are preferred by most others. Heuristic behavior with such a preference distribution is not analyzed yet. Evaluations in a real-world setting would be of a great interest and will be made in cooperation with the university hospital.

Acknowledgments

The authors would like to thank Sumedha Widyadharma for helping to develop the scheduler GUI, Martin Beer, Kai Jakobs, and the anonymous reviewers for their valuable comments and corrections.

References

1. P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers Group, PO Box 322, 3300 AH Dordrecht, The Netherlands, 2001.
2. M. Becker, K.-H. Krempels, M. Navarro, and A. Panchenko. Agent Based Scheduling of Operation Theatres. *EU-LAT eHealth Workshop, Cuernavaca, Mexico*, pages 220–227, December 2003.
3. Edmund Kieran Burke, Patrick De Causmaecker, and Greet Vanden Berghe. Novel Meta-heuristic Approaches to Nurse Rostering Problems in Belgian Hospitals. In *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 44.1–44.18. A CRC Press Company, 2004.
4. Stefan J. Darmoni, Alain Fajner, Nathalie Mahe, Arnaud Leforestier, Marc Von-dracek, Olivier Stelian, and Michel Baldenweck. Horoplan: computer-assisted nurse scheduling using constraint based programming. In *Journal of the Society for Health Systems*, volume 5, pages 41–54, 1995.

5. Michael R. Garey and David S. Johnson. *Computers and intractability: A Guide to the theory of NP-completeness*. A Series of Books in the Mathematical Sciences. Freeman and Co., San Francisco, California, USA, 1979.
6. Y. Guo, A. Lim, B. Rodrigues, and Y. Zhu. Heuristics for a Brokering Set Packing Problem. In *Proceedings of 8th International Symposium on Artificial Intelligence and Mathematics in Ft. Lauderdale, Florida*, 2004.
7. D. Hochbaum. *Approximation Algorithms for NP-hard problems*. PWS Publishing Company, 20 Park Plaza, Boston, MA 02116-4324, 1995.
8. Florian Kandler. Scheduling in a Virtual Enterprise in the Service Sector. In Juergen Sauer, editor, *Proceedings of the KI-2001 Workshop "AI in Planning, Scheduling, Configuration and Design", PuK 2001 Vienna, Austria, September 18, 2001*, pages 32–42, 2001.
9. Lars Kraglund and Torben Kabel. *Employee Timetabling: An Empirical Study of Solving Real Life Multi-Objective Combinatorial Optimization Problems by means of Constraint-Based Heuristic Search Methods*. Master thesis in cs, 1998.
10. Andriy Panchenko. *Preference-Based Scheduling of Operation Theaters*. Master Thesis in Computer Science, Department of Computer Science IV, RWTH Aachen University, Germany, 2005.
11. T.O. Paulussen, N.R. Jennings, K.S. Decker, and A. Heinzl. Distributed Patient Scheduling in Hospitals. In *Proceedings of 18th International Conference on AI, Acapulco, Mexico, 09.-15. August*, 2003.
12. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Upper Saddle River, New Jersey 07458, 2002.
13. D.G. Rajpathak. Intelligent Scheduling - A Literature Review. Technical Report KMI-TR-119, Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK, August 2001.
14. J. Sauer. *Multi-Site Scheduling: Hierarchisch koordinierte Ablaufplanung auf mehreren Ebenen*. PhD thesis, Fachbereich Informatik, Universität Oldenburg, 2002.

Artificial Immune Algorithms for University Timetabling

Muhammad Rozi Malim¹, Ahamad Tajudin Khader², and Adli Mustafa³

¹ Faculty of Info. Technology & Quantitative Sciences, UiTM, 40450 Shah Alam, Malaysia
rozi@tmsk.uitm.edu.my

² School of Computer Science, USM, 11800 Minden, Penang, Malaysia
tajudin@cs.usm.my

³ School of Mathematical Science, USM, 11800 Minden, Penang, Malaysia
adli@cs.usm.my

Abstract. The university timetabling, examination and course, are known to be highly constrained optimization problems. Metaheuristic approaches, and their hybrids, have successfully been applied to solve the problems. This paper presents three artificial immune algorithms, the algorithms inspired by the immune system, for university timetabling; clonal selection, immune network and negative selection. The main objective is to show that the algorithms may be tailored for educational timetabling. The experimental results have shown that all algorithms have effectively produced good quality timetables. The clonal selection and negative selection are more effective than immune network in producing good quality examination timetables; while for course timetabling, the immune network and negative selection are more effective than clonal selection. A comparison with other published results has significantly shown the effectiveness of these algorithms. The main operators in artificial immune algorithms are cloning and mutation. For future work, these algorithms will be improved by considering other cloning and mutation operators.

Keywords: Examination Timetabling; Course Timetabling; Artificial Immune Algorithms.

1 Introduction

The constructions of *examination* and *course* (lecture) timetables are common problems for all institutions of higher education. Usually it involves modifying the previous semester's timetable so it will work for the new semester. The examination and course timetabling are known to be highly constrained combinatorial optimization problems. Metaheuristic approaches such as simulated annealing (SA), tabu search (TS), evolutionary algorithms (EA), and their hybrids, have successfully been applied to solve the problems.

Artificial immune system (AIS), a new branch of Artificial Intelligence [4], is a new intelligent problem-solving technique that being used in optimization and scheduling. The AIS algorithms are more efficient than the classical heuristic scheduling algorithms such as SA, TS, and genetic algorithm (GA) [12]. AISs have

been more successful than GA and other methods in applications of pattern recognition, computer and network security, and dynamic tasks scheduling due to the applicability features of natural immune systems. Furthermore, the solutions produced by the AIS are observed to be *robust* than solutions produced by a GA [13]. There are *three* algorithms that have widely been applied in AIS; *clonal selection algorithm* (CSA), *immune network algorithm* (INA), and *negative selection algorithm* (NSA).

This paper presents three artificial immune algorithms for examination and course timetabling. The main objective is to show that the algorithms may be tailored for educational timetabling, and also to compare the effectiveness of the three algorithms on examination and course datasets. Twelve *Carter* datasets (examination) and three *Schaerf* datasets (course) have been used in the implementation. The experimental results have significantly shown the effectiveness of the three algorithms; all algorithms have effectively produced good quality (low fitness) examination and course timetables in most of the datasets. The CSA and NSA are more effective than INA on examination datasets, and on course datasets, the INA and NSA are more effective than CSA. However, based on CPU time, INA runs faster than CSA and NSA on examination datasets, and CSA runs faster than INA and NSA on course datasets. A comparison with other published results have significantly shown that the three algorithms are capable of producing good quality examination and course timetables as good as other optimization algorithms.

The main operators in artificial immune algorithms are *cloning* and *mutation*. For future work, these algorithms will be improved by considering other cloning and mutation operators so that the fitness values may be further minimized. And also, especially for course datasets, a further study is required to solve timetabling problems with 100% occupancy by considering dummy timeslots and/or rooms.

2 University Timetabling Problems

University timetabling problems can be divided into *two* main categories: exam and course. The main difference is that in course timetabling there cannot be more than one course per room, but in exam timetabling there can be more than one exam.

Examination timetabling problem (ETP) is a specific case of the more general timetabling problem. The examination timetabling regards the scheduling for the exams of a set of university courses, avoiding overlap of exams of courses having common students, and spreading the exams for the students as much as possible [7]. Given is a set of exams, a set of timeslots, a set of students, and a set of student enrollments to exams, the problem is to assign exams to timeslots subject to a variety of *hard* and *soft* constraints. The ETP can be seen as consisting of *two* subproblems: (1) assigning exams to timeslots, and (2) assigning exams to rooms. For real-life situations, these two subproblems can be solved separately.

Course timetabling problem (CTP) is another specific case of the more general timetabling problem. At its simplest, course timetabling is the problem of scheduling a set of events (lectures, tutorials or labs) to a set of classrooms in a set of timeslots within a week, and taught by a set of teachers, such that no student or teacher is expected to be in more than one room at the same time and that there is enough space in each classroom for the number of students expected to be there. The CTP can be

seen as consisting of *three* subproblems; ‘course-teacher assignment’, ‘event-timeslot assignment’, and ‘event-room assignment’. In ‘course-teacher assignment’, the teachers are scheduled to a number of events in all courses; in ‘event-timeslot assignment’, all events for all courses are scheduled into a fixed number of timeslots; and in ‘event-room assignment’, these events are assigned to a fixed number of rooms. Hence, an *assignment* is an ordered 4-tuple (a, b, c, d) , and has the straightforward general interpretation: ‘event a starts at timeslot b in room c , and is taught by teacher d ’. For some institutions, the allocation of courses to teachers is carried out manually, and the allocation of events in a given timeslot to rooms is a secondary problem. These three subproblems can be solved separately.

Hard constraints must be satisfied in order to produce a *feasible* timetable. Any timetable fails to satisfy these constraints is deemed to be *infeasible*. Soft constraints are generally more numerous and varied, and far more dependent on the needs of the individual problem than the more obvious hard constraints. The violation of soft constraints should be minimized; it is the soft constraints which effectively define how good a given feasible solution is so that different solutions can be compared and improved via an objective (*fitness*) function.

3 Artificial Immune System and Artificial Immune Algorithms

The ‘artificial immune system’ is an approach which used the natural immune system as a metaphor for solving computational problems, *not* modeling the immune system [21]. The main application domains of AIS are anomaly detection [16], pattern recognition [23], computer security [14], fault tolerance [1], dynamic environments [18], robotics [19], data mining [20], optimization [22], and scheduling [12].

The ‘immune system’ (IS) can be considered to be a remarkably efficient and powerful information processing system which operates in a highly parallel and distributed manner [11]. It contains a number of features which potentially can be adapted in computer systems; recognition, feature extraction, diversity, learning, memory, distributed detection, self-regulation, threshold mechanism, co-stimulation, dynamic protection, and probabilistic detection. It is unnecessary to replicate *all* of these aspects of the IS in a computer model, rather they should be used as general guidelines in designing a system.

There are a number of different algorithms that can be applied to many domains, from data analysis to autonomous navigation [5]. These immune algorithms were inspired by works on theoretical immunology and several processes that occur within the IS. The AISs lead to the development of different techniques, each one mapping a different mechanism of the system. For examples, the *Artificial Immune Networks* as proposed by Farmer *et al.* [9], the *Clonal Selection Algorithm* proposed by de Castro and Von Zuben [6], and the *Negative Selection Algorithm* introduced by Forrest *et al.* [10]. Immune network models are suitable to deal with dynamic environments and optimization problems, algorithms based upon the clonal selection principle are adequate to solve optimization and scheduling problems, and the negative selection strategies are successfully applied to anomaly detection.

3.1 Clonal Selection Algorithms for University Timetabling

The clonal selection algorithm (CSA) is inspired by the immunological processes of *clonal selection* and *affinity maturation*. When an antigen is detected, those antibodies that best recognize this antigen will proliferate by cloning. This process is called *clonal selection principle* [6]. The clonal selection principle is used to explain how the IS ‘fights’ against an antigen. When a bacterium invades our organism, it starts multiplying and damaging our cells. One form the IS found to cope with this replicating antigen was by replicating the immune cells successful in recognizing and fighting against this disease-causing element. Those cells reproduce themselves asexually in a way proportional to their degree of recognition: the better the antigenic recognition, the higher the number of clones (offspring) generated. During the process of cell division (reproduction), individual cells suffer a mutation that allows them to become more adapted to the antigen recognized: the higher the affinity of the parent cell, the lower the mutation they suffer. Figure 1 shows the CSA for exam or course.

```

1. Initialization: initialize a population of antibodies (feasible timetables)
for each antibody (timetable)
    randomly select event (exam/course) one by one
        assign event to random selected timeslots and rooms (satisfying hard constraints)
        if no identical antibodies (duplicate timetables)
            add antibody (timetable) to the population
        else eliminate antibody
2. Population loop: for each generation of antibodies (feasible timetables)
for each antibody do
    determine the affinity of antibody via an affinity function (affinity = 1/fitness)
    calculate the selection probability (rate of cloning) using affinity
        (selection probability = affinity/total affinities)
    randomly select an antibody (timetable) based on selection probability
        (using roulette wheel selection method)
2.3 Genetic variation: Cloning: clone copies of the selected antibody
(number of clones = population size × cumulative selection probability)
Mutation: for each generated clone, do (mutation rate = 1 - selection probability)
if a random probability <= mutation rate, mutation = failure
    while mutation = failure, select an event at random
        reassigned event to random timeslot and room (satisfying all hard constraints)
if all hard constraints are satisfied and no duplicate timetables
    determine the affinity of the new clone
    if the affinity (new clone) >= the affinity (original clone)
        mutation = success
2.4 Population update: if the affinity (new clone) > minimum affinity (population), say antibody X
    then X = new clone
3. Cycle: repeat (Step 2) until stopping criteria are met.

```

Fig. 1. Clonal Selection Algorithm for University Timetabling

The main operators in CSA are *selection*, *cloning*, and *mutation*. A timetable (high affinity) is randomly selected for cloning using Roulette Wheel selection method and, on average, a number of clones that equal to half of the population size are generated. Almost all clones will be mutated to produce new feasible timetables for the next generation since ‘1- selection probability’ would give a high mutation rate for each clone. But only new timetables with high affinity will be selected to replace the low affinity timetables in the current population. The process (selection, cloning and mutation) will be repeated until the stopping criteria are met.

3.2 Immune Network Algorithms for University Timetabling

The immune network algorithm (INA) is based on *Jerne's idiotypic network theory* [15]. According to this theory, immune cells have portions of their receptor molecules that can be recognized by other immune cells in a way similar to the recognition of an invading antigen. This results in a network of recognition between immune cells. When an immune cell recognizes an antigen or another immune cell, it is stimulated. On the other hand, when an immune cell is recognized by another immune cell, it is suppressed. The sum of the stimulation and suppression received by the network cells, plus the stimulation by the recognition of an antigen corresponds to the stimulation level S of a cell. Figure 2 shows the INA for examination or course timetabling.

```

1. Initialization: initialize a network (population) of immune cells (feasible timetables)
for each immune cell (timetable)
randomly select event one by one
    assign event to random timeslot and room (satisfying all hard constraints)
if no identical immune cells (duplicate timetables)
    add immune cell to the population
else eliminate immune cell
2. Population loop: for each network (generation/population) of immune cells (feasible timetables)
2.1 Network interactions and Stimulation:
for each immune cell
determine the fitness of immune cell via a fitness function
calculate the stimulation level of immune cell (stimulation level = 1/fitness)
determine the total stimulation of the network (population)
calculate the stimulation probability for each immune cell
(stimulation probability = stimulation/total stimulation)
2.2 Metadynamics (Antigens and Genetic variations):
for each immune cell
cloning – generate a number of clones
(number of clones = population size × stimulation probability)
for each clone
determine the mutation rate (mutation rate = 1 – stimulation probability)
generate a random probability
if a random probability <= mutation rate
    mutation = failure
    while mutation = failure
        select an event at random
        reassign event to random timeslot and best room
        if all hard constraints are satisfied and no duplicate timetables
            mutation = success
            determine the fitness of the new clone
            if the fitness (new clone) > the fitness (original clone)
                mutation = failure, and reset the reassignment
            else (no mutation) assign a zero stimulation (large fitness) to immune cell
2.3 Network dynamics (immune cells and antigens interactions, and population update):
gather all immune cells (current population and cloned timetables)
sort immune cells according to stimulation level (descending order)
select the best (high stimulation) immune cells (feasible timetables)
(number of selected immune cells = network or population size)
update the network (population) of immune cells with the selected cells
3. Cycle: repeat Step 2 until a given convergence or stopping criterion is met.

```

Fig. 2. Immune Network Algorithm for University Timetabling

The main operators in INA are *cloning* and *mutation*. All timetables are selected for cloning and, on average, one clone is generated for each timetable. Almost all clones will be mutated to produce new feasible timetables since ‘1- stimulation probability’ would give a high mutation rate for each clone. All feasible timetables, current population and mutated clones, are gathered, but only the timetables with high stimulation will be selected to form a new population for the next generation. The process (cloning and mutation) will be repeated until the stopping criteria are met.

3.3 Negative Selection Algorithms for University Timetabling

The negative selection algorithm (NSA) is the most widely used techniques in AISs. The NSA is based on the principles of self-nonself discrimination [10]. The algorithm was inspired by the thymic negative selection process that intrinsic to natural immune systems, consisting of screening and deleting self-reactive T-cells, i.e. those T-cells that recognize self cells. Figure 3 shows the NSA for examination or course timetabling.

```

1. Initialization: initialize a population of candidate detectors (initial feasible timetables)
   for each candidate detector (timetable)
      randomly select event one by one
         assign event to random timeslot and room (satisfying all hard constraints)
         if no identical candidate detectors (duplicate timetables)
            add candidate detector to the initial population
         else eliminate candidate detector
2. Population loop: for each generation (population) of detectors (feasible timetables)
3.1 Censoring: for each detector (timetable) in the current population
   determine the fitness value via a fitness function (soft constraints)
   determine the average fitness for the current population
   for each detector
      if the fitness >= average, eliminate the detector
      if all fitness values are equal, eliminate only the second half of the detectors
3.2 Monitoring: while the number of detectors (timetables) < population size
   randomly select a detector according to fitness using roulette wheel
   clone the detector, mutation = failure
   while mutation = failure, randomly select an event
      reassign event to random timeslot and best room
      if all hard constraints are satisfied and no identical detectors
         mutation = success
         determine the fitness of new clone
         if the fitness of the new clone > average fitness of the population
            mutation = failure
            eliminate the new clone, and reset the reassignment
         else add the new clone to the new population
3. Cycle: repeat population loop until a given convergence criterion is met.

```

Fig. 3. Negative Selection Algorithm for University Timetabling

The main operators in NSA are *negative deletion* (censoring), *cloning* and *mutation*. The timetables (current population) with fitness greater than or equal to average fitness are eliminated or deleted from the current population. A timetable is randomly selected from the remaining timetables for cloning and mutation using Roulette Wheel selection method (based on fitness). All clones will be mutated to produce new feasible timetables. For each new (mutated) timetable, if the fitness is less than or equal to average, the timetable will be added to the new population for the next generation; otherwise, it will be deleted. The monitoring process (cloning and mutation) will be repeated until the number of feasible timetables in the new population is equal to population size. Finally, the optimization process (censoring and monitoring) will be repeated until the stopping criteria are met.

4 Benchmark Datasets

The benchmark datasets (*Carter* and *Schaerf*) used in the implementation of the *three* immune algorithms are available from <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/> and <http://www.diegm.uniud.it/schaerf/projects/coursestt/>, respectively. These datasets provide reasonable benchmark problems for comparison of the three different artificial immune algorithms. The datasets are shown in Table 1 and Table 2.

Table 1. Examination Datasets and Characteristics (Carter Datasets)

Code	University	No. of Exams	No. of Students	No. of Enrollments	Timeslot Capacity
car-f-92	Carleton University 1992	543	18419	55522	2000
car-s-91	Carleton University 1991	682	16925	56877	1550
ear-f-83	Earl Haig Collegiate 1983	190	1125	8109	350
hec-s-92	Ecole des Hautes Etudes Comm 92	81	2823	10632	650
kfu-s-93	King Fahd University 1993	461	5349	25113	1955
lse-f-91	London Sch. of Econ. 1991	381	2726	10918	635
rye-s-93	Ryerson University 1993	486	11483	45051	2055
sta-f-83	St. Andrews High 1983	139	611	5751	465
tre-s-92	Trent University 1992	261	4360	14901	655
uta-s-92	Uni. of Toronto, Arts & Science 92	622	21266	58979	2800
ute-s-92	Uni. of Toronto, Engineering 92	184	2750	11793	1240
yor-f-83	York Mills Collegiate 1983	181	941	6034	300

Each of the datasets come in two files, one file (*course data file*) contains the list of courses and the other (*student data file*) contains a list of student-course selections. The courses and student-course selections are sorted in ascending order.

Table 2. Course Datasets and Characteristics (Schaerf Datasets)

Instance	No. of Courses	No. of Rooms (R)	No. of Timeslots (T)	Timeslots per day	Total lectures (L)	No. of Teachers	Occupancy (L/(R×T))
1	46	12	20	4	207	39	86.25%
2	52	12	20	4	223	49	92.92%
3	56	13	20	4	252	51	96.92%
4	55	10	25	5	250	51	100%

Each of the datasets comes in *five* files; *course.dat* contains the information about the courses, *periods.dat* contains the list of timeslots of the timetabling horizon, *curricula.dat* contains the information about groups of courses that share common students, *constraint.dat* contains additional constraints about timeslot unavailabilities, and *room.dat* contains information about rooms. The ‘occupancy’ indicates the percentage of timeslot-room required to schedule all the lectures.

However, the dataset ‘Instance 4’ was not considered; 100% occupancy would make the mutation process impossible, and perhaps *dummy* timeslots and rooms would solve the problem. This requires more time and further study, and will be included in the future work.

5 Comparing Artificial Immune Algorithms on Exam Datasets

The three artificial immune algorithms (CSA, INA, NSA) have been implemented on the twelve examination datasets (Carter datasets). The main objective is to compare the effectiveness of the three algorithms on examination datasets.

Three hard constraints were considered for each of the datasets: (1) no students must be assigned to two different exams at the same timeslot, (2) timeslot capacity must not be exceeded, and (3) each exam must be assigned to exactly one timeslot. The *fitness value* (soft constraint violations) is the minimum number of students having two exams in adjacent (consecutive) timeslots.

The following (Table 3) are the experimental results for examination datasets using the three artificial immune algorithms. Each algorithm was run on each dataset for five trials, and the maximum number of generations ‘500’ was used as the stopping criterion. The best fitness, the average fitness and the average CPU time (in seconds) for each algorithm on each of the datasets, based on five trials, have been recorded.

Table 3. Comparing Three Artificial Immune Algorithms on Examination Datasets

Institution	No. of Exams	No. of Timeslots	Fitness Values					
			CSA		INA		NSA	
			Best	Average Fitness (CPU time)	Best	Average Fitness (CPU time)	Best	Average Fitness (CPU time)
car-f-92	543	31	285	466.6 (310.6s)	406	455.2 (249.8s)	386	432.8 (359.4s)
car-s-91	682	40	535	569.6 (512.6s)	554	582.8 (399.6s)	439	486.2 (484s)
ear-f-83	190	24	17	48 (75.4s)	65	112.8 (34.8s)	74	118.8 (88s)
hec-s-92	81	19	3	11 (17.2s)	0 (271)	9.8 (7.8s)	5	14.4 (11.4s)
kfu-s-93	461	20	35	69.4 (172.4s)	16	32.6 (202.2s)	2	13.6 (240.2s)
lse-f-91	381	18	45	68.8 (132.4s)	34	82.6 (120.8s)	115	167.2 (147s)
rye-s-93	486	24	143	240.2 (233.8s)	217	309 (247s)	180	327.6 (336.4s)
sta-f-83	139	14	0 (196)	0 (12.2s)	0 (185)	0.4 (11.4s)	0 (160)	0 (9.6s)
tre-s-92	261	25	27	36.8 (110s)	58	70.2 (57.4s)	56	79.2 (134s)
uta-s-92	622	32	436	487.6 (343.2s)	374	436 (307.4s)	165	244.6 (387s)
ute-s-92	184	10	0 (352)	0.4 (34.8s)	0 (454)	2.6 (26.8s)	1	9.8 (34.4s)
yor-f-83	181	22	3	8 (62.6s)	24	33.2 (30.4s)	1	6.6 (63.2s)

The *number of timeslots* used for all datasets were imposed according to those given in Carter's results. However, the number of timeslots for all datasets may be further reduced if necessary. Based on five trials, for the *best fitness*, both CSA and NSA have achieved the first position in *five* datasets, while INA has achieved the first position in only *two* datasets. The best fitness values for CSA have converged to '0' in *two* datasets, INA in *three* datasets, and NSA in *one* dataset. For the *average fitness*, both CSA and NSA has achieved the first position in *six* datasets, and INA in only *one* dataset. Finally, for the *average CPU time*, INA has achieved the first position in *nine* datasets, only *two* for CSA and *one* for NSA.

Hence, it may be concluded that CSA and NSA are equally effective in producing good quality (low fitness) examination timetables, and both are more effective than NSA. Based on CPU time, INA runs faster than CSA and NSA. The results from 12 different examination datasets have significantly shown the effectiveness of the three algorithms. All algorithms have effectively produced good quality examination timetables in most of the datasets.

A comparison with other published results was also conducted. This is to access the effectiveness of the three algorithms against other optimization algorithms. Only *five* datasets were considered; car-f-92, car-s-91, kfu-s-93, tre-s-92, and uta-s-92. The following are the authors and metaheuristic approaches used in the published results:

- (A) Burke *et al.* [2] – Memetic Algorithm.
- (B) Di Gaspero and Schaerf [7] – Tabu Search.
- (C) Caramia *et al.* [3] – A set of heuristics: Greedy Assignment, Spreading Heuristic.
- (D) Merlot *et al.* [17] – Hybrid Algorithm: Constraint Programming, Simulated Annealing, Hill-climbing.

All authors had considered the same hard and soft constraints, hence a comparison based on the fitness values (number of students having two exams in adjacent timeslots) may be carried out. The main goal is to show that the immune algorithms can produce good quality examination timetables as good as other methods. The number of timeslots used for all datasets were imposed according to the papers of the published results. The maximum number of none-improvement generations '25' was considered as the stopping criterion. Table 4 summarizes the results.

Table 4. Comparison with Other Solution Methods

Code	Timeslots (Sessions)	Timeslot Capacity	Fitness Values						
			A	B	C	D	Average Fitness		
							CSA	INA	NSA
car-f-92	31	2000	331	424	268	158	75.7	97.3	154.3
car-s-91	51	1550	81	88	74	31	33.0	73.7	21.7
kfu-s-93	20	1995	974	512	912	247	5.3	12.0	5.0
tre-s-92	35	655	3	4	2	0	7.7	13.0	8.0
uta-s-92	38	2800	772	554	680	334	24.7	81.7	12.7

The results of other solution methods are available on Internet from <http://www.or.ms.unimelb.edu.au/timetabling/ttframe.html?ttexp3.html>. Hence, based on five trials, and five datasets, the three artificial immune algorithms have effectively produced good quality examination timetables, as good as other solution methods.

6 Comparing Artificial Immune Algorithms on Course Datasets

The three artificial immune algorithms (CSA, INA, NSA) have been implemented on the three course datasets (Schaerf datasets). The main objective is to compare the effectiveness of the algorithms on course datasets.

Five hard constraints were considered for all datasets: (1) all lectures of all courses must be scheduled, (2) two distinct lectures cannot take place in the same room in the same timeslot, (3) lectures of courses of the same group must be all scheduled at different timeslots, (4) lectures of courses taught by the same teacher must be scheduled at different timeslots, and (5) lectures of some courses must not be assigned to certain timeslots. The *fitness value* (soft constraint violations) is the number of students without a seat, plus the number of courses that assigned to less than the minimum number of days multiply by 5, and plus the number of gaps between lectures of the same group on the same day multiply by 2.

The following (Table 5) are the experimental results for course datasets using the three artificial immune algorithms. Each algorithm was run on each dataset for five trials, and the maximum number of generations 1000 was used as the stopping criterion. The best fitness, the average fitness, and the average CPU time (in seconds) for each algorithm on each of the datasets, based on five trials, have been recorded.

Table 5. Comparing Three Artificial Immune Algorithms on Course Datasets

Instance	No. of Courses	Fitness Values									Di Gaspero & Schaerf (2003)	
		CSA			INA			NSA				
		Best	Ave	Ave CPU	Best	Ave	Ave CPU	Best	Ave	Ave CPU		
1	46	284	297	1560s	265	296	3976s	263	298	1583s	200	
2	52	21	46	347s	11	21	1190s	21	36	525s	13	
3	56	69	98	1617s	50	72	5873s	48	74	2254s	55	

The results have significantly shown the effectiveness of the three algorithms. All algorithms have effectively produced good quality course timetables with low fitness values in all datasets. For the *best fitness*, NSA has achieved the first position in *two* datasets, while INA in *one* dataset. For the *average fitness*, INA has achieved the first position in *all* datasets. Finally, for the *average CPU time*, CSA has achieved the first position in *all* datasets. It may be concluded that, based on three datasets and five trials, NSA and INA are more effective than CSA in producing good quality course timetables; however, CSA runs faster than INA and NSA.

There is only one published result available on these datasets, by Di Gaspero and Schaerf [8], as shown on the right-hand side of Table 5; available from <http://www.diegm.uniud.it/projects/EduTT/>. The artificial immune algorithms have achieved the first position in two datasets (Instances 2 and 3). However, no results have been produced by artificial immune algorithms for Instance 4; 100% occupancy in Instance 4 requires dummy timeslots and/or rooms for the mutation process. This requires more effort and time and will be considered in the future work.

7 Conclusion and Future Work

This paper has presented and compared three artificial immune algorithms for the university timetabling problems; CSA, INA and NSA. The experimental results using twelve Carter datasets (examination) and three Schaerf datasets (course) have significantly shown the effectiveness of these algorithms on university timetabling datasets. All algorithms have efficiently produced good quality examination and course timetables with low fitness values in most of the datasets. For examination datasets, CSA and NSA are both more effective than INA in producing good quality timetables; while for course datasets, NSA and INA are more effective than CSA. Based on CPU time, INA runs faster than CSA and NSA on examination datasets, and CSA runs faster than INA and NSA on course datasets.

All artificial immune algorithms show great promise in the area of educational timetabling, particularly in its ability to consider, solve, and optimize the wide variety of different examination and course timetabling problems. The algorithms can handle the hard constraints and soft constraints very well. The experimental results have shown that the algorithms can successfully be applied to solve various kinds of examination and course timetabling problems. These algorithms may be accepted as new members of evolutionary algorithms for solving timetabling problems.

The most important operators in artificial immune algorithms are cloning and mutation. For future work, these algorithms will be improved by considering other operators, especially mutation, so that the fitness values may be further minimized. However, different timetabling problems may require different operators. A good cloning or mutation operator for one problem is not necessarily a good operator for other problems. For the course timetabling, the three algorithms were not designed to handle timetabling problems with 100% occupancy as in Instance 4 of Schaerf datasets. Perhaps the use of dummy timeslots and/or rooms will solve the problems. This will be the first priority in our future research.

References

1. Bradley, D.W., and Tyrrell, A.M.: Immunotronics: Hardware Fault Tolerance Inspired by the Immune System. Proceedings of the International Conference on Evolvable Systems (ICES2000). April 17-19, Edinburgh, UK (2000) 11-20.
2. Burke, E.K., Newell, J.P., and Weare, R.F.: A memetic algorithm for university exam timetabling. Proceedings of the First International Conference on Practice and Theory of Automated Timetabling (ICPTAT 1995). August 30 – Sept 1, Edinburgh, UK (1995) 496-503.
3. Caramia, M., Dell'Olmo, P., and Italiano, G.F.: New algorithms for examination timetabling. Lecture Notes in Computer Science 1982, S. Nher and D. Wagner (eds), Springer-Verlag, Berlin-Heidelberg, Germany, (2001) 230.
4. Dasgupta, D., Ji, Z., and González, F.: Artificial Immune System (AIS) Research in the Last Five Years. Proceedings of the International Conference on Evolutionary Computation (CEC2003). December 8-12, Canberra, Australia (2003).

5. de Castro, Leandro Nunes: Immune, Swarm, and Evolutionary Algorithms, Part I: Basic Models. Proceeding of the ICONIP, Workshop on Artificial Immune Systems, Vol. 3. November 18-22, Singapore (2002) 1464-1468.
6. de Castro, L.N. and Von Zuben, F.J.: The Clonal Selection Algorithm with Engineering Applications. Proceedings of The Genetic and Evolutionary Computation Conference 2000 (GECCO'OO) - Workshop Proceedings. July 8-12, Las Vegas, USA (2000) 36-37.
7. Di Gaspero, L. and Schaerf, A.: Tabu search techniques for examination timetabling. In Practice and Theory of Automated Timetabling, Third International Conference, E.K. Burke and W. Erben (eds): Lecture Notes in Computer Science 2079, Springer-Verlag, Konstanz, Germany (2001) 104-117.
8. Di Gaspero, L. and Schaerf, A.: Multi Neighborhood Local Search with application to the Course Timetabling Problem. Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT2002). Lecture Notes in Computer Science 2740. Springer-Verlag, Berlin-Heidelberg, Germany (2003).
9. Farmer, J.D., Packard, N.H. and Perelson, A.S.: The immune system, adaptation, and machine learning. *Physica*, 22D (1986) 187-204.
10. Forrest, S., Perelson, A.S., Allen, L. and Cherukuri, R.: Self-nonself discrimination in a computer. Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy Los Alamitos, CA: IEEE Computer Society Press (1994) 202-212.
11. Hart, Emma: Immunology as a Metaphor for Computational Information Processing: Fact or Fiction? PhD Thesis, University of Edinburgh (2002).
12. Hart, E. and Ross, P.: An Immune System Approach to Scheduling in Changing Environments. Proceedings of The Genetic and Evolutionary Computation Conference 1999 (GECCO'99). July 13-17, Florida, USA (1999) 1559-1566.
13. Hart, E., Ross, P., and Nelson, J.: Producing Robust Schedules Via An Artificial Immune System. Proceedings of the International Conference on Electronic Commerce 1998 (ICEC'98). April 6-9, Seoul, Korea (1998).
14. Hofmeyr, S.A. and Forrest, S.: Architecture for an Artificial Immune System. Research Notes. Department of Computer Science, University of New Mexico (2003).
15. Jerne, N.K.: Towards a Network Theory of the Immune System. *Ann. Immunol. (Inst. Pasteur)* 125C (1974) 373-389.
16. Kim, J., Ong, A., and Overill, R.: Design of an Artificial Immune System as a Novel Anomaly Detector for Combating Financial Fraud in Retail Sector. Congress on Evolutionary Computation 2003 (CEC2003), Dec 8-12, Canberra (2003).
17. Merlot, L.T.G., Boland, N., Hughes, B.D. and Stuckey, P.J.: A Hybrid Algorithm for the Examination Timetabling Problem. Lecture Notes in Computer Science 2740, E.K. Burke and P. De Causmaecker (eds), Springer-Verlag, Belgium (2003) 207.
18. Simões, A., and Costa, E.: An Immune System-Based Genetic Algorithm to Deal with Dynamic Environments: Diversity and Memory. Proceedings of the 6th International Conference on Neural Networks & Genetic Algorithms. April 23-25, France (2003).
19. Singh, S., and Thayer, S.: Immunology Directed Methods for Distributed Robotics: A Novel, Immunity-Based Architecture for Robust Control & Coordination. Proceedings of SPIE: Mobile Robots XVI, Vol. 4573 (2001).
20. Timmis, Jonathan: Artificial Immune Systems: A Novel Data Analysis Technique Inspired by the Immune Network Theory. Dissertation, University of Wales (2000).
21. Timmis, Jonathan: An Introduction to Artificial Immune Systems. Research Notes, ES2001, Cambridge, University of Kent at Canterbury, England, UK (2001).
22. Walker, J.H. and Garrett, S.M.: Dynamic Function Optimization: Comparing the Performance of Clonal Selection and Evolution Strategies. Proceeding of Second International Conference on Artificial Immune Systems (ICARIS2003). September 1-3, Napier University, Edinburgh, UK (2003).
23. White, J.A. and Garrett, S.M.: Improved Pattern Recognition with Artificial Clonal Selection? Proceeding of Second International Conference on Artificial Immune Systems (ICARIS2003). September 1-3, Napier University, Edinburgh, UK (2003).

An Empirical Investigation on Memes, Self-generation and Nurse Rostering

Ender Özcan

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

Abstract. In this paper, an empirical study on self-generating multimeme memetic algorithms is presented. A set of well known benchmark functions is used during the experiments. Moreover, a heuristic template is introduced for solving timetabling problems. The heuristics designed based on this template can utilize a set of constraint-based hill climbers in a cooperative manner. Two such adaptive heuristics are described. Memetic algorithms utilizing each one as if a single hill climber are experimented on a set of random nurse rostering problem instances. Additionally, simple 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), introduced by J. Holland [27], are very promising for tackling complex problems [24]. Effectiveness of hill climbing methods in population based algorithms is underlined by many researchers [15, 38, 45, 46]. Memetic Algorithms (MAs) embody a class of algorithms that combine genetic algorithms and hill climbing methods. A *meme* represents a hill climbing method and its related parameters used within an MA. Ning et al. [39] concluded from their experiments that the meme choice in an MA influence the performance significantly. Krasnogor [31] extended the previous studies and suggested a *self-generating (co-evolving) multimeme* MA for solving problems in the existence of a set of hill climbers. Memes are evolved with the candidate solutions, providing a learning mechanism to fully utilize the provided hill climbers [32, 34].

In the first part of this study, the MA 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 et al. [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.

2 Background

2.1 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]. Only difference is that the noise component of the Quartic function is modified as described in [53].

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

<i>label</i>	<i>function name</i>	<i>lb</i>	<i>ub</i>	<i>opt</i>	<i>source</i>
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	0	[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]

Eight memes are used in the experiments:

- Steepest Descent (MA0), [37]
- Next Descent (MA1), [37]
- Random Mutation Hill Climbing (MA2), [37]
- Davis's Bit Hill Climbing (MA3), [16]

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 represent candidate solutions during benchmark experiments for continuous and discrete functions, respectively. Due to the gray encoding, the memes MA4-MA7 represent *poor* hill climbers for almost all continuous benchmark functions.

2.2 Nurse Rostering Problem

Timetabling problems are real-world constraint optimization problems. Due to their NP complete nature [20], 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:

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

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:

$$V' = \{v'_1 = v_1, \dots, v'_i = v_i, \dots, v'_M = v_M\}, \text{ where } v'_i \in d_i \text{ and } d_i \subseteq D_1 \times \dots \times D_P, \text{ where } P \geq 1$$

In all timetabling problems, there is at least one domain for each variable that is for time. A problem instance might require other resources to be scheduled as well. For example, a university course timetabling problem instance might require arrangement of 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 employees, employers, even 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 *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*) [42]. Edge constraints are the most common constraints that represent pairs of variables to be scheduled without a clash. A timetabling problem reduces to graph coloring 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 neighborhood search using a set of different perturbation methods and local search algorithms on randomly generated schedules. Chun et al. [13] modeled nurse rostering as 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] modeled nurse rostering as a weighted constraint satisfaction problem. Their algorithm consists of two phases. In the first phase, forward checking, variable ordering and compulsory backjumping are used, whereas in the second phase descend local search and tabu search are used. Ahmad et al. [1] proposed a population-less cooperative genetic algorithm and experimented on a 3-shift problem. Kawanaka et al. [30] attempted to meet absolute and desirable constraints fro obtaining optimal nurse schedules. Aickelin et al. proposed a co-evolutionary pyramidal GA and experimented an indirect representation and three different decoders within GA for solving NRP in [3], [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 multiobjective approach, prioritizing the objectives. Heuristic swaps working and rest days. Duenas et al. [19] applied interactive Sequential Multiobjective 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] introduced 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 GA that they were developed for solving 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.

2.3 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 opera-

tors 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. After the crossover, a set of new individuals, called *offspring*, is generated. Offspring are then *mutated*. In MAs, 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. Whenever the *termination criteria* are satisfied, 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-heuristics are discussed in [11].

Multimeme Algorithms (MMAs) represent a subset of self generating (co-evolving) MAs [31-34]. An individual in a population carries a memetic material along with a genetic material. 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 favors 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 as Lamarckian learning mechanism [31, 40].

Using a similar notation as provided in [33], a *meme*, denoted by $M_{hFBbInWt}$, represents the hill climbing method (M), its acceptance strategy (FB), the maximum number of iterations (I), and which part of the configuration to apply the selected method (W). 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 favored. The steeper the slope gets for a meme, the more it is favored.

3 Memetic Algorithms for Benchmarking

3.1 Experimental Setup

All runs are repeated 50 times. Pentium IV 2 GHz. machines with 256 MB RAM are used during the experiments. 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 sec., 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; $t=\{\text{whole}\}$.

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

<i>label</i>	<i>dim.</i>	<i>no. of bits</i>	<i>chrom. length</i>	<i>pop. size</i>	<i>max. hc steps</i>
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

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; $h=\{\text{MA0}, \text{MA1}, \text{MA2}, \text{MA3}\}$ 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 as GA, MA0-MA3.

3.2 Empirical Results for the Benchmark Functions

Performance comparison of genetic algorithm and memetic algorithms using different memes are presented in Fig. 1 for selected benchmark functions based on the average number of evaluations. For each experiment, related bar appears in the figure, only if all the runs yield the expected result. 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.

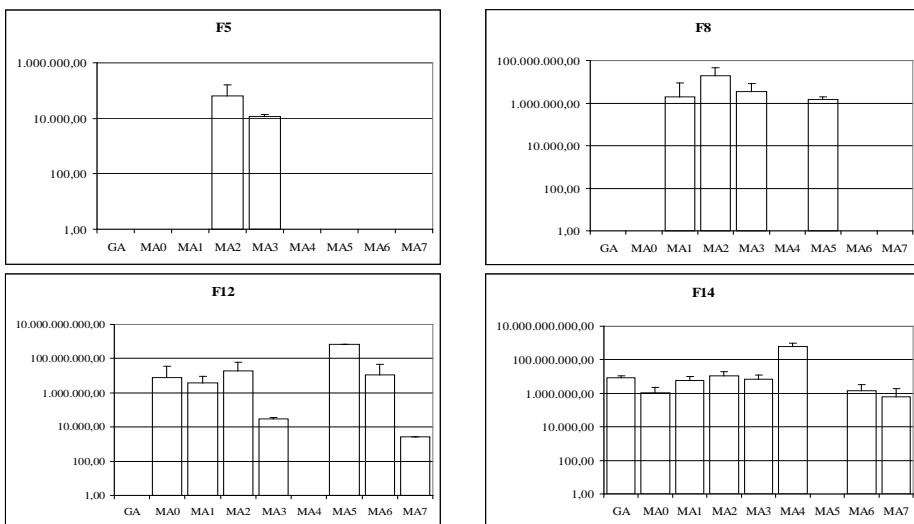


Fig. 1. Mean and the standard deviation of the number of evaluations per run, generated by each MA for a selected subset of benchmark functions

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 Fig. 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.

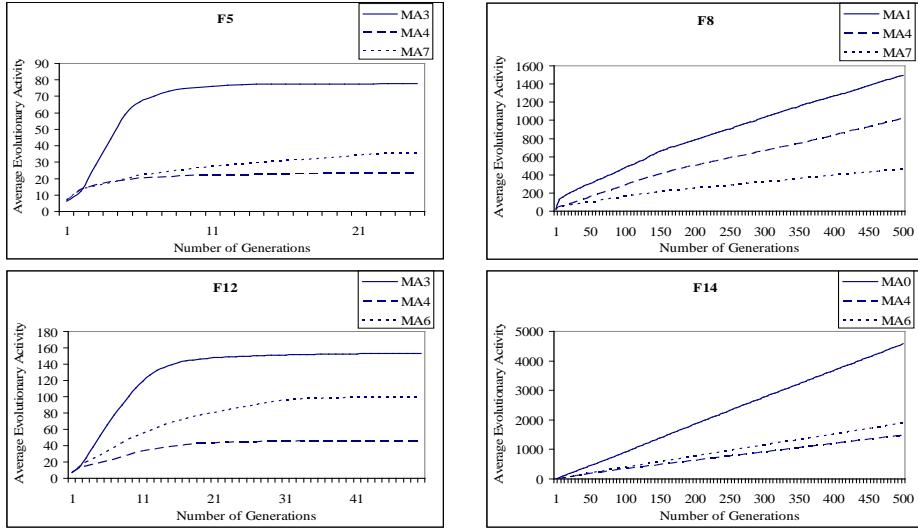
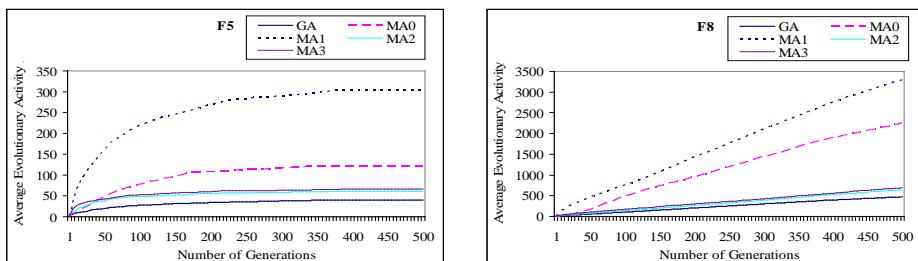


Fig. 2. Average evolutionary activity vs. generation plots of each meme utilized during the second set of experiments for a selected subset of benchmark functions

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 illustrated in Fig. 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 indicate that the MA with the best meme is superior based on the average number of evaluations, except for F1, F3 and F11 (Table 3).



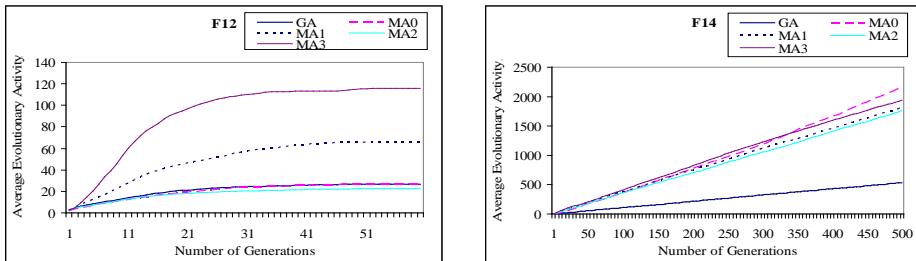


Fig. 3. Average evolutionary activity vs. generation plots of each meme utilized during the third set of experiments for a selected subset of benchmark functions

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

		avr. no. of evals.	st.dev.			avr. no. of evals.	st.dev.
label	type			label	type		
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

4 Memetic Algorithms for Nurse Rostering

4.1 Nurse Rostering Problem at the Memorial Hospital (NRPmh)

An analysis is performed on the Nurse Rostering Problem at the Memorial Hospital (NRPmh), located in Istanbul, Turkey. 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 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;

Excludes:

- Exclude Night Shifts Constraint (ENC): Night shifts can not 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 can not be assigned to more than two successive night shifts.
- Successive Day Shifts Constraint (SDC): A nurse can not be assigned to more than three successive day shifts.
- Successive Shifts Constraint (SSC): A nurse can not 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.
- On-duty Constraint (ODC): Each nurse can not 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

Ozcan [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 Fig. 4. The VTDHC supports adaptation and cooperation of operators. It is a more general template than the 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.

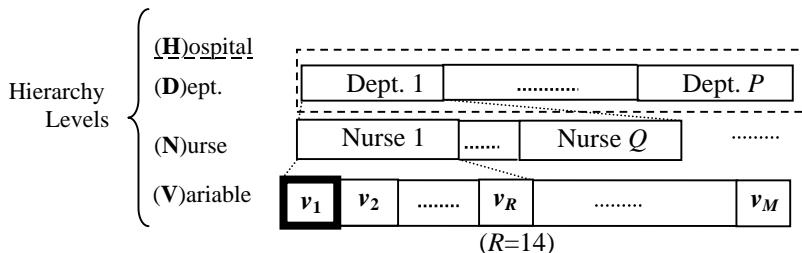
```

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 the VTDHC

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

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

Combining the arrangements and VTDHC yields the design of useful hyper-

heuristics. For example, VDHC represents a subset of 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 the 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. The 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 the VDHC is adaptive and in a way self-adjusting. During the traversal of an arrangement downwards in the hierarchy levels, the VDHC switches from individual level adaptation to 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 multiobjective problems. Moreover, heuristics based on the 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 determining which 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 $l=T*14$, where l is chromosome length. The search space size for finding the optimal schedule becomes immense; 3^l . The traditional approaches fail to obtain a solution, 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 Fig. 5.

Seven hill climbing (HC) operators are designed to be used in the 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 $\{\text{whole}, \text{department}, \text{nurse}\}$. 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 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. The MMA experiments using this operator are performed for three different IR values.

During the second set of experiments, hierarchical traversal of groups is reversed in the 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, the r VDHC stays at the nurse level. A selected hill climbing method is applied in the same way as the VDHC as described in Section 4.2. The r VDHC 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 r VDHC and NHC are hyper-heuristics that are instances of VTDHC.

In the fourth set of experiments, a multimeme algorithm is implemented. MMA uses 7 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. 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/~eozcan/research/TTML>.

Table 4. Experimental data set, where the number of departments and nurses are denoted as n_{dep} and n_{nur} , 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	n_{dep}	n_{nur}	$pnr0$	$pnr1$	$pnr2$	$avrpr$
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

5.2 Empirical Results for the NRP Experiments

Detailed experimental results of the MA with the VDHC are presented in [41]. The results obtained from the first set of experiments indicate the viability of 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 MMA selects mostly a nurse roster and then applies a hill climber to it, as illustrated in Fig. 6 for IR=0.20. The rest of the experiments are performed on Pentium IV 3 GHz. machines with 2 GB RAM.

Table 5. MMA 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

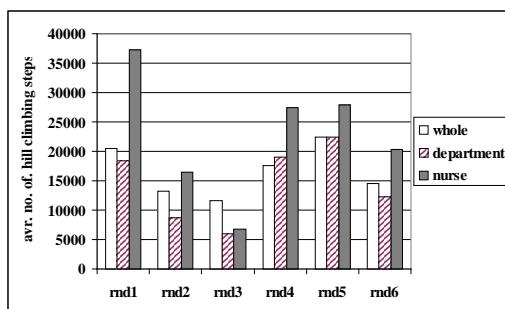


Fig. 6. Average number of hill climbing steps that are executed to improve the whole set of daily shifts, a departmental roster and a nurse roster for each problem instance during the first set of experiments, where IR=0.20

During the preceding sets of experiments, the MAs with *rVDHC*, *NHC*, the simple genetic algorithm and the *MMA7* are tested on the problem instances. The success rate of each algorithm for each problem instance is presented in Table 6. Obviously, hill climbing boosts the performance GAs. Simple genetic algorithm turns out to be the worst algorithm for solving the problem instances. Almost; in none of the runs a violation free schedule is obtained. Empirical results yield the success of MAs with the following hill climbers from the best towards the worst: *VDHC*, *rVDHC* and *NHC*, respectively. The average performance of *MMA7* is comparable to the performance of *NHC*. Results show that letting the multimeme algorithm to choose the region where to apply a constraint based hill climber based on a static hierarchical arrangement of events performs better than to let it to choose which meme to use for solving nurse rostering problem instances.

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

Label	VDHC	rVDHC	NHC	MMA7	Simple GA
rnd1	0.96	0.94	0.68	0.86	0.00
rnd2	1.00	0.98	0.88	0.96	0.04
rnd3	1.00	1.00	0.98	1.00	0.00
rnd4	0.98	0.94	0.28	0.18	0.00
rnd5	1.00	0.86	0.26	0.30	0.00
rnd6	1.00	1.00	0.68	0.50	0.00

6 Conclusions

Memetic algorithms, including the self-generating multimeme memetic algorithm proposed by Krasnogor [33] are investigated. Different MAs are experimented using a set of benchmark functions and nurse rostering problem instances, generated randomly by Ozcan [41] based on NRPmh. Some common empirical results are ob-

tained from both investigations. As expected, the performance of a genetic algorithm improves if a hill climbing operator is also utilized. Lamarckian learning mechanism employed by the MMAs yields appealing results for selecting a meme among a set of memes during the evolutionary process. Yet, the MAs with a good meme choice perform better. Different memes yield different performances. In the benchmark experiments, the MMAs identify the useful memes for all functions, but unfortunately, a synergy between hill climbers is not observed during the search. The average performance of the Davis's Bit Hill Climbing is the best on the benchmark functions.

The MAs are very promising approaches for tackling nurse rostering problems. 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. The MAs, each containing such an instance as a single hill climber are compared to the MMAs, with different memetic materials. The empirical results indicate the success of the MA with VDHC [41] over the rest of the MAs presented in this paper. The VDHC using tournament selection provides a better cooperation 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, the *r*VDHC shows potential.

Acknowledgement. This research is supported 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. *Genetic Algorithms and Simulated Annealing*, (1987) 170-215
2. Ahmad, J., Yamamoto, M., and Ohuchi, A.: Evolutionary Algorithms for Nurse Scheduling Problem. *Proc. of IEEE Congress on Evolutionary Computation* (2000) 196-203.
3. Aickelin, U., and Bull, L.: On the Application of Hierarchical Coevolutionary Genetic Algorithms: Recombination and Evaluation Partners. *JASS*, 4(2) (2003) 2-17
4. Aickelin, U., and Dowsland, K.: An Indirect Genetic Algorithm for a Nurse Scheduling Problem. *Computers & Operations Research*, 31(5) (2003) 761-778
5. Alkan, A., and Ozcan, E.: Memetic Algorithms for Timetabling. *Proc. of IEEE Congress on Evolutionary Computation* (2003) 1796-1802
6. Berrada, I., Ferland, J., and Michelon, P.: A Multi-Objective Approach to Nurse Scheduling with both Hard and Soft Constraints. *Socio-Economic Planning Science*. v1. 30(1996)183-193
7. Burke, E.K., Cowling, P.I., De Causmaecker, P., and Vanden Berghe, G.: A Memetic Approach to the Nurse Rostering Problem, *Applied Intelligence*, vol 15 (2001) 199-214
8. Burke, E.K., De Causmaecker, P., Petrovic, S., Vanden Berghe G.: Variable Neighbourhood Search for Nurse Rostering Problems, in *Metaheuristics: Computer Decision-Making* (edited by M.G.C. Resende and J. P. de Sousa), Chapter 7, Kluwer (2003) 153-172

9. Burke, E.K., De Causmaecker, P., and Vanden Berghe, G.: A Hybrid Tabu Search Algorithm For the Nurse Rostering Problem, Proc. of the Second Asia-Pasific Conference on Simulated Evolution and Learning, vol. 1, Applications IV (1998) 187-194
10. Burke, E.K., De Causmaecker, P., and Vanden Berghe, G., Van Landeghem, H.: The State of the Art of Nurse Rostering, Journal of Scheduling, 7 (2004) 441-499
11. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S.: Handbook of metaheuristics, chapter 16, Hyper-heuristics: an emerging direction in modern search technology, Kluwer Academic Publisher (2003) 457-474
12. Burke, E., and Soubeiga, E.: Scheduling Nurses Using a Tabu-Search Hyper-heuristic, Proc. of the 1st MISTA, vol. 1 (2003) 197-218
13. Chun, A.H.W., Chan, S.H.C., Lam, G.P.S., Tsang, F.M.F., Wong, J., and Yeung, D.W.M.: Nurse Rostering at the Hospital Authority of Hong Kong, Proc. of 17th National Conference on AAAI and 12th Conference on IAAI (2000) 951-956
14. Cowling P., Kendall G., and Soubeiga E.: A Hyper-heuristic Approach to Scheduling a Sales Summit. Proceedings of In LNCS 2079, Practice and Theory of Automated Timetabling III : Third International Conference, PATAT 2000, Konstanz, Germany, selected papers (eds Burke E.K. and Erben W) (2000) 176-190
15. Davis, L.: The handbook of Genetic Algorithms, Van Nostrand Reingold, NY (1991)
16. Davis, L.: Bit Climbing, Representational Bias, and Test Suite Design, Proceeding of the 4th International conference on Genetic Algorithms (1991) 18-23
17. De Jong, K.: An analysis of the behaviour of a class of genetic adaptive systems. PhD thesis, University of Michigan (1975)
18. Downshall, K.: Nurse Scheduling with Tabu Search and Strategic Oscillation, European Journal of Operations Research. Vol. 106, 1198 (1998) 393-407
19. Duenas, A., Mort, N., Reeves, C., and Petrovic, D.: Handling Preferences Using Genetic Algorithms for the Nurse Scheduling Problem, Proc.of the 1st MISTA, vol.1(2003)180-196
20. Easom, E. E.: A survey of global optimization techniques. M. Eng. thesis, Univ. Louisville, Louisville, KY (1990)
21. Even, S., Itai, A., and Shamir, A.: On the Complexity of Timetable and Multicommodity Flow Problems, SIAM J. Comput., 5(4) (1976) 691-703
22. Fang, H.L. Genetic Algorithms in Timetabling and Scheduling, PhD thesis, Department of Artificial Intelligence, University of Edinburgh, Scotland (1994)
23. Gendreau, M., Buzon, I., Lapierre, S., Sadr, J., and Soriano, P.: A Tabu Search Heuristic to Generate Shift Schedules, Proc. of the 1st MISTA, vol.2 (2003) 526-528
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 (1989) 129-152
26. Goldberg, D. E.: Genetic algorithms and Walsh functions: part II, deception and its analysis, Complex Systems (1989) 153-171
27. Griewank, A.O.: Generalized descent of global optimization. Journal of Optimization Theory and Applications (1981) 34: 11.39
28. Holland, J. H.: Adaptation in Natural and Artificial Systems, Univ. Mich. Press (1975)
29. Han, L., and Kendall, G.: Application of Genetic Algorithm Based Hyper-heuristic to Personnel Scheduling Problems, Proc. of the 1st MISTA, vol.2 (2003) 528-537
30. Kawanaka, H., Yamamoto, K., Yoshikawa, T., Shinogi, T., and Tsuruoka, S.: Genetic Algorithms with the Constraints for Nurse Scheduling Problem, Proc. of IEEE Congress on Evolutionary Computation (CEC), Seoul (2001) 1123-1130
31. Krasnogor, N.: Studies on the Theory and Design Space of Memetic Algorithms, PhD Thesis, University of the West of England, Bristol, United Kingdom (2002)

32. Krasnogor, N. and Smith, J.E.: Multimeme Algorithms for the Structure Prediction and Structure Comparison of Proteins. In Proc. of the Bird of a Feather Workshops, GECCO (2002) 42-44
33. Krasnogor, N. and Smith, J.E.: Emergence of Profitable Search strategies Based on a Simple Inheritance Mechanism. In Proc. of the Genetic and Evolutionary Computation Conference, GECCO (2001) 432-439.
34. Krasnogor, N. and Smith, J.E.: A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study. In Proc. of the Genetic and Evolutionary Computation Conference, GECCO (2000) 987-994.
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., and Rodrigues, B.: A Hybrid AI Approach for Nurse Rostering Problem, Proc. of the 2003 ACM Symposium on Applied Computing (2003) 730-735
37. Mitchell M., Forrest S.: Fitness Landscapes: Royal Road Functions, *Handbook of Evolutionary Computation*, Baeck T, Fogel D, Michalewicz Z (Ed.), Institute of Physics Publishing and Oxford Univers (1997)
38. Moscato, P., and Norman, M. G.: A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems, *Parallel Computing and Transputer Applications* (1992) 177-186
39. Ning, Z., Ong, Y. S., Wong, K. W. and Lim, M. H.: Choice of Memes In Memetic Algorithm, Proc. of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems (2003)
40. Ong, Y.S. and Keane, A.J.: Meta-Lamarckian Learning in Memetic Algorithms. *IEEE Trans. Evolutionary Computation*, vol. 8, no. 2 (2004) 99-110
41. Ozcan, E.: Memetic Algorithms for Nurse Rostering. *Lecture Notes in Computer Science*. Springer-Verlag, The 20th ISCIS (2005) 482-492
42. Ozcan, E.: Towards an XML based standard for Timetabling Problems: TTML, *Multidisciplinary Scheduling: Theory and Applications*, Springer Verlag (2005) 163 (24)
43. Ozcan, E., and Alkan, A.: Timetabling using a Steady State Genetic Algorithm, *Proceedings of the 4th PATAT* (2002) 104-107
44. Ozcan, E., Ersoy, E.: Final Exam Scheduler - FES, Proc. of 2005 IEEE Congress on Evolutionary Computation, vol. 2, (2005) 1356-1363
45. Ozcan, E., and Onbasioglu E.: Genetic Algorithms for Parallel Code Optimization, Proc. of 2004 IEEE Congress on Evolutionary Computation, vol. 2 (2004) 1775-1781
46. Radcliffe, N. J., and Surry, P.D.: Formal memetic algorithms, *Evolutionary Computing: AISB Workshop*, LNCS, vol. 865, Springer Verlag (1994) 1-16
47. Rastrigin L. A.: Extremal control systems. In *Theoretical Foundations of Engineering Cybernetics Series*. Moscow: Nauka, Russia. (1974)
48. Ross, P., Corne, D., and Fang, H-L.: Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation, Proc. of PPSN III (1994) 556-565
49. Ross, P., Corne, D., and Fang, H-L.: Fast Practical Evolutionary Timetabling, Proc. of AISB Workshop on Evolutionary Computation (1994) 250-263
50. Schwefel, H.-P.: Numerical optimization of computer models. Chichester: Wiley & Sons. (1981)
51. Schwefel, H. P.: *Evolution and Optimum Seeking*. John Wiley & Sons. (1995)
52. Smith, J. and Fogarty, T. C.: Operator and parameter adaptation in genetic algorithms. *Soft Computing* 1(2): 81-87 (1997)
53. Tasoulis D., Pavlidis N., Plagianakos V, Vrahatis M.: Parallel Differential Evolution. Proc. of 2004 IEEE Congress on Evolutionary Computation (2004) 2023-2029
54. Whitley, D.: Fundamental principles of deception in genetic search. In G.J.E. Rawlins (Ed.), *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Matco, CA (1991)

Solving the University Timetabling Problem with Optimized Enrolment of Students by a Parallel 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 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 problem. Comparing to previous approaches we designed the encoding for self-adaptation not only one parameter or several ones but for all possible parameters of genetic algorithms at the same time. Genetic algorithms are naturally parallel so also the parallel representation of the self-adaptive genetic algorithm is presented. The proposed parallel 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 possible to solve 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 also we present and discuss the algorithm for optimized enrolment of students that minimize 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 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 genetic algorithm optimizes parameters of the second-level. A self-adaptation seems to be a promising way of genetic algorithms, 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 and genetic opera-

tors for self-adaptation of genetic algorithms that is suitable for solving the university timetabling problem. Comparing to previous approaches (e.g. [2], [12], [20]) we designed the encoding for self-adaptation 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. Genetic algorithms are naturally parallel so also the parallel representation of the self-adaptive genetic algorithm is presented.

The proposed parallel 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 it is known no algorithm for solving it in polynomial time [8]. 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, even it is difficult to find only 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 decrease 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 [19]. 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 [23]. 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 [5, 11]. 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 [1, 7, 14, 18, 21, 22].

When applying genetic algorithms to some optimization or scheduling problem, the crucial element is encoding. For timetabling problem there are two main types of encoding: direct [18] and indirect [14]. The advantage of direct encoding is that the whole search space can be encoded, but it usually leads to violation of many hard constraints. The 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 limited portion of the search space and thus it can be trapped in a local optimum. In our application some combination of both encoding was used,

because we want to let the algorithm decide itself whether to use the direct or indirect representation, respectively 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 parallel self-adaptive genetic algorithm is compared to standard genetic algorithms on this timetabling problem. Also the role of enrollment optimization algorithm is discussed.

2 Encoding

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 behind the proposed encoding consists in redundancy of information through hierarchical evaluation of individuals.

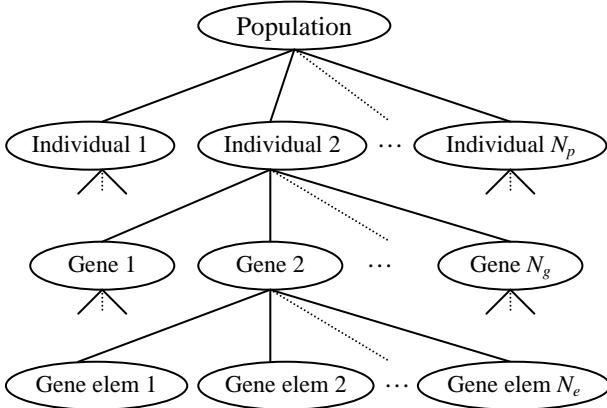


Fig. 1. The structure of a population

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 evolution process, but only indirectly by evaluation mechanism from low level values.

Table 1. The structure of a gene element

Name	Description	Range
x^E	Optimized variable	<0;1>
q_m^E	Parameter of mutation	<-1;1>
q_p^E	Parameter of protected mutation	<-1;1>
r_m^E	Radius of mutation	<0;0.5>
p_c^E	Probability of crossover	<0;1>
r_c^E	Ratio of crossover	<0;1>
q_d^E	Parameter of deletion	<-0.1;0.1>
q_u^E	Parameter of duplication	<-0.1;0.1>
q_t^E	Parameter of translocation	<-0.1;0.1>
s_m^E	Identifier of myself for mating	<0;1>
s_w^E	Wanted partner for mating	<0;1>
r_r^E	Ratio of replacement	<0;1>
r_t^E	Ratio of population for selection	<0;1>
r_p^E	Ratio of population for 2 nd partner selection	<0;1>
c_d^E	Coefficient of death	<0;1>
N_p^E	Wanted size of population	<0;1>
i_t^E	Identifier of translocation	<0;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 simple arithmetical average, i.e.

$$X^G = \frac{1}{N_e} \sum_{i=1}^{N_e} X_i^E, \quad (1)$$

where X stands for parameters that must be evaluated, i.e. $x, s_m, s_w, r_r, r_t, r_p, c_d, N_p, i_t$.

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.

$$X^I = \frac{1}{N_g} \sum_{i=1}^{N_g} X_i^G \quad (2)$$

The number of genes N_g is not variable, because one gene contains exactly one optimized variable.

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

$$w_f = \frac{\frac{N_p^P - i + 1}{(1 + N_p^P)N_p^P}}{2} \quad (3)$$

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.

3 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 following paragraphs.

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. Important is that he cannot take into account his genotype, i.e. directly the values of his genes nor his fitness, but only his phenotype, i.e. only 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 individual's preferences for mating and the parameter s_m^I represents 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 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 limited ratio of population r_p^I .

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 formula

$$X_3^E = X_1^E + (X_2^E - X_1^E) \cdot r_c^E \quad (4)$$

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.

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

$$X_{new}^E = X_{old}^E + (X_{max}^E - X_{min}^E) \cdot U(-r_m^E, r_m^E) \quad (5)$$

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_{new}^E is the value of the parameter after mutation, X_{old}^E is the original value of the parameter, X_{max}^E (X_{min}^E) is the maximal (minimal) allowed bit element value of the parameter as defined in Table 1.

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, 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.

Deletion

The deletion operator is applied to every gene element with probability $p_d^E = |q_d^E|$. It means that the gene element is simply removed from the particular gene. By deletion and duplication operators the degree of polyploidy is controlled.

Translocation

The 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

$$p = p_t^E \cdot \left(1 - \left|i_t^{G(\text{new})} - i_t^{G(\text{old})}\right|\right) \quad (6)$$

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 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 .

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 previous than the mutated chromosome is used otherwise the old chromosome is used for following evolution cycle.

Replacement of Individuals

For every individual the parameter of a 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

$$c_L = 1 - c_d^P \left(1 - w_f\right) \quad (7)$$

Evidently, through evolution, a less fitter individual causes the 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.

4 The Parallel Self-adaptive Genetic Algorithm

Genetic algorithms are naturally parallel so it invokes an idea to implement the proposed self-adaptive genetic algorithm parallelly. We used simple parallel structure, in which there is one master genetic algorithm and N_a slave genetic algorithms. The master genetic algorithm is responsible for performing genetic operators and creating new generation, while the slave genetic algorithm just calculate the value of fitness function of the individual presented by the master genetic algorithm. In each generation the master genetic algorithm distributes N_p/N_a tasks to each slave genetic algorithm. The master genetic algorithm communicates with slave genetic algorithms by TCP/IP protocol. The structure of the parallel self-adaptive genetic algorithm is depicted in Figure 2.

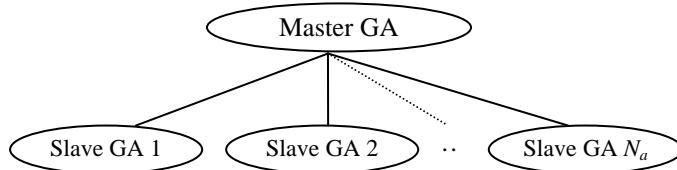


Fig. 2. The structure of the parallel self-adaptive genetic algorithm

5 The Timetabling Problem

In this chapter 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 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 is 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 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 requirements for suitable room is handled as the 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 were satisfied and the number of soft constraint violations was minimal. This problem can be mathematically formulated as optimization model minimizing error of the timetable defined as

$$z = \sum_{i=1}^{n_L} \left[\sum_{j=1}^{n_G} x_{ij} p_{ij} + \sum_{k=1}^{n_L} c_{ik} \text{sametime}(i, k) \right] + w_p \cdot \text{penalty}(x_{ij}) \rightarrow \min \quad (8)$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{i=1}^{n_L} x_{ij} \leq 1 \text{ for } j = 1, 2, \dots, n_G \\ & \sum_{j=1}^{n_G} x_{ij} = 1 \text{ for } i = 1, 2, \dots, n_L, \end{aligned}$$

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 penalty of the timetable that is not possible to express by clash matrix C or preference matrix P . And the coefficient w_p is the weight of $\text{penalty}(x_{ij})$ by which it contributes to the error of the timetable.

6 Teacher Preferences

To express teacher preferences we define following criteria. For each criterion the teacher sets its preference t^* by “mark” from 1 to 5, the 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 if $t^* = 1$ it means the best possibility for the teacher and actually no error of timetable, we transform t^* to t^{**} by decreasing 1, i.e. $t^{**} = t^* - 1$. If the preference $t^* = 5$ it is hard constraint, so $t^{**} = 10^6$.

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 time-room 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 could look like that:

	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

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 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 in 2 or 3 days per week, in 4 days it is not convenient for him, in 5 days it is not acceptable for him and in 1 day it is not possible to teach all lectures then the preferences t_k^N could look like that:

Number of Teaching Days	1	2	3	4	5
Preferences t_k^N	5	1	1	3	5

Length of Teaching Block without Break

By this criterion the teacher sets if he prefers to concentrate lectures to one long teaching block or to disperse it 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 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 would not like to have too dispersed lectures, i.e. he wants to teach at least 2 hours without break, the most 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 could look like that:

The 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

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 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 for the teacher it is very inconvenient 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 6 hours per day is impossible then the preferences t_k^H could look like that:

The Number of Hours	1	2	3	4	5	6	7	8	9	10
Preferences t_k^H	5	4	2	1	1	2	3	5	5	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.

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.

7 Enrollment of Students

At most universities there are some group 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, even it

is hardly to find only 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 of subjects consist of two kinds of lectures: 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 appropriate seminary for each student. One possibility is to do it randomly, but by this way it will be very difficult or nearly impossible to build the timetable, in which each student has unclashing timetable or the number of clashing lectures for students is acceptably small. So we propose the algorithm for optimized enrolment of students that minimize the number of clashing constraints for students.

In the model we use the following notation

- S_{ij}^U – binary variable defining whether the student S_i is enrolled to the subject U_j
- S_{ij}^L – binary variable defining whether the student S_i is enrolled to the lecture L_j
- U_{ij}^L – binary variable defining whether the subject U_i contains the lecture L_j
- S_{ij}^L – binary variable defining whether the student S_i is enrolled to the lecture L_j
- L_i^S – maximal number of students that can be enrolled to the lecture L_i

Without loss of generality suppose that each kind of lecture of the same subject will be labeled as different subject. First we set the elements c_{ij}^T of clash matrix C for teacher clashes as was described in the chapter 5. After that student 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 the lecture to which the student should be enrolled. The core of the enrollment problem is then to enroll all students to all lectures such that all constraints were satisfied and the number of nonzero elements c_{ij} of the clash matrix C was minimal. The problem can be mathematically formulated as optimization model minimizing the number of nonzero elements c_{ij} defined as

$$\begin{aligned}
 c &= \sum_{i=1}^{n_L} \sum_{j=1}^{n_L} \text{nonzero}(c_{ij}) \rightarrow \min && (9) \\
 \text{s.t.} \quad & \sum_{i=1}^{n_S} S_{ij}^L \leq L_j^S \quad \text{for } j = 1, 2, \dots, n_L \\
 & \sum_{j=1}^{n_U} S_{ij}^U = \sum_{k=1}^{n_L} S_{ik}^L \quad \text{for } i = 1, 2, \dots, n_S \\
 & \sum_{j=1}^{n_L} S_{ij}^L \cdot U_{kj}^L \cdot S_{ik}^U = 1 \quad \text{for } i = 1, 2, \dots, n_S, k = 1, 2, \dots, n_U
 \end{aligned}$$

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 the function which is equal to 0 when c_{ij} is zero and 1 otherwise.

8 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.

The Enrolment Builder

First we enroll students to lectures according their preferences for subjects they want to attend by the process described in the chapter 7. For optimization of enrollment of students the proposed parallel self-adaptive genetic algorithm was used. As was mentioned in the chapter 2, each gene of a chromosome represents one real variable within the interval $<0;1>$. 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 the part A of the chromosome and B_j is the j -th parameter of the part B of the chromosome. The fitness function f for the genetic algorithm is the negative value of c in (9), i.e. $c = -z$.

The Timetable Builder

For solving the university timetabling problem the parallel 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 values of parameters in the 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 the part A of the chromosome and B_j is the j -th parameter of the 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 the formula (8). 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 (8), i.e. $f = -z$.

To make the idea behind decoding the chromosome more clear, a simple example will be provided. Let's 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 8 genes. Let's suppose that after evaluation, the gene values of parameters x^E 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^E , 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	0,03	0,10	0,17
L1	0,37	0,09	0,04	0,11
L2	0,09	0,55	0,42	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 look like that: L1→G3, L2→G1, L3→G2.

9 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 (8), 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 parallel 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. Previously used approach for constructing the timetable produced the timetable, in which there were in average 2.8 clashing lectures for each student, moreover it was very boring and time consuming process, because the timetable was completely made manually, computer was used only as graphical user interface.

In order to test also 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 we can see 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.

We also tested the role of enrollment optimization algorithm. The enrolment optimization algorithm as described in the chapter 7 was substituted by random enrollment students to lectures and the best solution found by the parallel SAGA was the timetable with the minimal value of error function $z = 12553$. As we can see it is much worse than with applying the enrollment optimization algorithm.

10 Conclusions

In this paper we have designed the 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 parallel 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 parallel 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.

Great problem has appeared when it was applied to the real timetabling problem with changed preferences and requirements for timetable, because the new timetable completely different comparing to the original one. So the further study will be concerned to deal with the problem of minimization of number of changes between new and original timetable.

References

1. Abramson, D. *Constructing school timetables using simulated annealing: sequential and parallel algorithms* European Journal of Operational Research. Management Science, vol. 37, no. 1, January 1991, pp. 98-113.
2. Bäck, T. *Self-Adaptation in Genetic Algorithms*. Proceedings of the First European Conference on Artificial Life. Cambridge: MIT Press, 1992.
3. Bufé, M. et al. *Automated Solution of a Highly Constrained School Timetabling Problem – Preliminary Results*. In Proceedings of the Evo Workshops 2001, Como, Italy: Springer, 2001.
4. Burke, E., Newall, J. Enhancing timetable solutions with local search methods. Lectures Notes in Computer Science 2740, pp. 195-206, Berlin: Springer – Velag, 2003.

5. Brailsford, S. C. et al. *Constraint Satisfaction Problems: Algorithms and Applications*. European Journal of Operational Research, vol 119, 1999, pp. 557-581.
6. De Jong, K. A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD. Thesis, University of Michigan, 1975.
7. Di Gaspero, L., Schaerf, A. *Tabu search techniques for examination timetabling*. Lectures Notes in Computer Science 2079, pp. 104-117, Berlin: Springer – Verlag, 2001.
8. Even, S., Iati, A., Shamir, A. *On the Complexity of Timetabling and Multicommodity Flow Problems*. Siam Journal of Computation, vol. 5, no. 4, pp. 691-703. 1976.
9. Fernandes, C. et al. *High School Weekly Timetabling by Evolutionary Algorithms*. In Proceedings of 14th Annual ACM Symposium on Applied Computing. San Antonio, Texas, 1999.
10. Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Massachusetts: Addison Wesley Readings, 1989.
11. Legierski, W. *Search Strategy for Constraint-Based Class-Teacher Timetabling*. Lectures Notes in Computer Science 2740, pp. 247-261, Berlin: Springer – Verlag, 2003.
12. Marsili, S. L., Alba, P. A. *Adaptive Mutation in Genetic Algorithms*. Soft Computing, vol. 4, no. 2, pp. 76-80. New York: Springer – Verlag, 2000.
13. Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd ed. New York: Springer – Verlag, 1996.
14. Paechter, B. et al. *Timetabling the Classes of an Entire University with an Evolutionary Algorithm*. Parallel Problem Solving from Nature (PPSN) V. Lectures Notes in Computer Science 1498, pp. 865-874, Berlin: Springer – Verlag, 1998.
15. Perzina, R. *Self-adaptation in Genetic Algorithms*. In Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), pp. 234-238. Orlando, FL, USA: IIIS, 2003.
16. Perzina, R. *A Self-adapting Genetic Algorithm for Solving the University Timetabling Problem*. In: Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), pp. 284-288. Orlando, USA: IIIS, 2004.
17. Perzina, R., Ramík, J. *A New Portfolio Selection Model Solved by Genetic Algorithms*. Proceedings of 20th International Conference MME 2002, pp. 201-207. Ostrava: VŠB TU, 2002.
18. Ross, P., Hart, E., Corne, D. *Some Observations about GA-based Exam Timetabling*. In The Practice and Theory of Automated Timetabling: Selected Papers from the Second International Conference. Lecture Notes in Computer Science 1408, pp. 115-129, Berlin: Springer – Verlag, 1997.
19. Schaerf, A. *A Survey of Automated Timetabling*. Artificial Intelligence Review, vol. 13, no. 2, pp. 87-127, Norwell: Kluwer Academic Publishers, 1999.
20. Stefano, C. D., Tettamanzi, A. G. B. *An Evolutionary Algorithm for Solving the Schoool Timetabling Problem*. In Proceedings of the Evo Workshops 2001, pp. 452-462, Como, Italy: Springer, 2001.
21. Terashima, H. et al. *Evolution of Constraint Satisfaction Strategies in Examination Timetabling*. In Proceedings of the Genetic and Evolutionary Computation Conference GECCO 1999, pp. 635-642, Morgan Kaumann, 1999.

22. Thompson, J.M., Dowsland, K.A. *A robust simulated annealing based examination timetabling system*. In Computers and Operations Research, vol. 25, pp. 637-648, Oxford: Elsevier Science Ltd., 1998.
23. White, G. M., Chan, P. W. *Towards the Construction of Optimal Examination Timetables*. INFOR 17, 1979, p.p. 219-229.
24. Yang, J. M., Kao, C. Y. *Integrating Adaptive Mutations and Family Competition into Genetic Algorithms as Function Optimizer*. Soft Computing, vol. 4, no. 2, pp. 89-102. New York: Springer – Verlag, 2000.

An Extensible Modelling Framework for the Examination Timetabling Problem

David Ranson¹ and Samad Ahmadi²

¹ Representational Systems Lab, Department of Informatics,
University of Sussex, Falmer, UK
d.j.ranson@sussex.ac.uk

² School of Computing, De Montfort University,
The Gateway, Leicester, LE1 9BH, UK
sahmadi@dmu.ac.uk

Abstract. A number of modelling languages for timetabling have been proposed to standardise the specification of problems, solutions and their data formats. These languages have not been adopted as standard due to not simplifying the modelling process, lack of 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 our experience of modelling the examination timetabling problem (ETP) using STTL. This framework is a work in progress but demonstrates the possibilities and convenience such a model would afford.

1 Introduction

In this paper, the rationale for proposing a new modelling framework for the ETP is discussed in relation to existing languages designed for timetabling. The timetabling problem itself is described followed by a brief survey of the existing languages. A model for the ETP 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.

Timetabling can be described as the general problem of “sequencing events subject to various constraints” [1]. This is typically, as the name suggests, assigning timeslots to events in order to create a feasible solution for a given problem. This is a complex task and the general timetabling problem is known to be NP-Hard.

Examination timetabling problem (ETP) is a significant special case of the general timetabling problem. Production of exam timetables is a practical challenge faced by almost all academic institutions on at least one occasion every year. The most important characteristics of the exam timetabling problem are the constraints that describe the problem.

The most important constraint violation 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 it may not

be violated in finding a feasible solution. Other examples of hard constraints are duration and room capacity constraints; e.g. exams cannot be scheduled into time periods with durations shorter than that of the exam.

The “consecutive exams” constraint is an example of a *soft* constraint. A violation of this constraint exists when a student is timetabled to sit more than one exam in immediate succession. This constraint exists in most instances of the exam timetabling problem, but may not be universal. Institutions may also add their own unique constraints such as not mixing 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 have some flexibility in the constraints which are specified.

The goal in exam timetabling is to minimize the number of violations of these constraints over a solution. Normally a cost is assigned to each type of constraint, with the hard constraints having much higher associated costs than the soft constraints. The total cost for a solution is then given as the sum of the costs for all the violations found.

There are many different and varying approaches to solving the exam timetabling problem being used at institutions and by researchers. A recent survey shows that these approaches include Sequential methods, Clustering approaches, Case-based reasoning and a number of Heuristic approaches[3]. This wide variety of the algorithms and software applications use different models and data formats adding to the cost of implementation due to handling of the model and the data.

The data published by Carter [4] (and other publicly available data) has been used for some benchmarking but 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.

The need for a modelling standard and a standard data format has been recognised 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 have lead to advances in research. Examples of this in practice are the Travelling Salesman Problem (TSPLIB) [6, 7] and the MPL (Mathematical programming language), MPS (Mathematical programming standard).

2 Progress Towards a Standard Format

There have been at least three attempts at creating modelling languages and standard data formats for timetabling problems since the proposal by Burke, Kingston and Pepper in 1998 [5]. These are the: Standard TimeTabling Language (STTL) [8, 9], TimeTabling Mark-up Language (TTML)[10] and UniLang [11].

STTL is a complete object oriented functional language designed to be suitable for modelling timetabling problems using set theory. STTL specifies the problem being modelled as well as the evaluation function for the model, instance data and solutions.

TTML is based on MathML which is an XML application for modelling maths formulae. The goal was to create a language with the functionality of STTL but using the fashionable XML. The TTML learning curve is steeper than that of STTL and again seems overly complicated, especially for specifying the complex logic involved in these problems.

UniLang is another language with 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 their design. In the first aim it has largely been superseded by 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 known exception to this is the publicly available datasets on the University of Melbourne Timetabling Problem Database website[4].

Perhaps, the main reason these languages have not been adopted as standard is that they offer no advantages to the user over any traditional programming language. These idealistic languages do not simplify the modelling process, and can even be restrictive in that they do not have all the features of a modern programming language, are overly complicated or appear cumbersome.

3 A Case Study: Modelling the Exam Timetabling Problem in STTL

An STTL model for the exam timetabling problem has been created and used as the data format for a working application[12, 13]. This model was based on the model Kingston [8] has created for the High School timetabling problem.

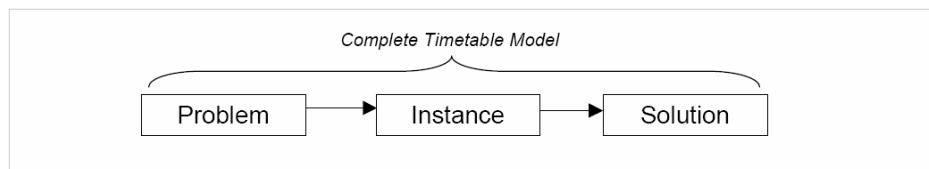


Fig. 1. The components of an STTL Model. A complete model is made up of the Problem, Instances of that problem and finally Solutions for the Instances

Each STTL problem is made up of three components, normally split into three different files. As its name suggests the Problem file contains the STTL code for modelling the problem, the constraints, and the evaluation function. The Instance file contains concrete data for instantiating a particular instance of the Problem and finally the Solution file contains values for the *solution variables* found in the Problem.

To model this problem we need to specify a problem file. The following two class diagrams show how we will model the entities and constraints found in our Exam Timetabling model:

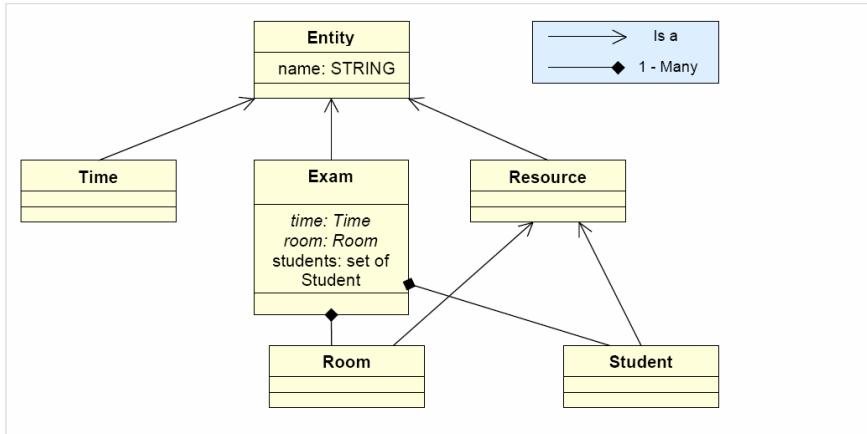


Fig. 2. The classes that make up our ETP model

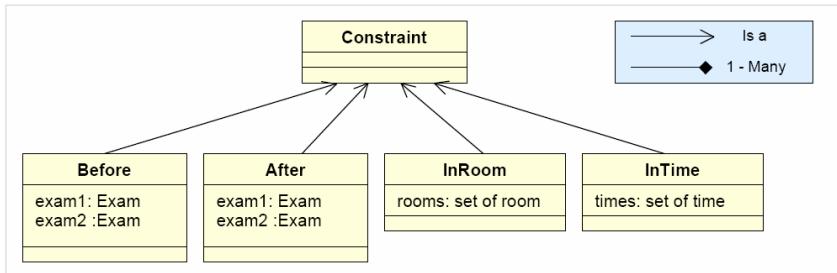


Fig. 3. The Constraints that are added to our ETP problem as Classes extending the Constraint class

This problem has been fully coded in STTL and is available for inspection and use online at: <http://www.informatics.sussex.ac.uk/users/djr23/STTL/>

The evaluation function can be used to evaluate existing solutions (in STTL format) demonstrating the functionality of STTL using the publicly available interpreter.

4 Limitations of this Model

This model was successfully used within a timetabling application[13], with existing data being converted to the STTL Instance file format. However this experience made

apparent some limitations of the model due to both our design approach and issues with STTL itself.

The STTL language involves a learning curve and, probably because of its nature as an Object Oriented Functional language, is quite complicated to use. Although set theory is one good way of specifying these kinds of problems it might not be the best way from a purely modelling point of view. The code fragment below, part of the STTL ETP evaluation function, illustrates the complexity of this language.

```
violations:SEQ[Violation] = (createViolation
  (roomViolationExist, name + " should not be scheduled
   in this room") + createVioion(timeViolationExist, name
   + " should not be scheduled in this time") +
  createViolation(clashExist(all Exam), "Room Clash in "+
  room.name + " at "+ time.name))
```

The design also introduces other complexities, distinct from those created due to the syntax; in the model presented above time is represented as a “Time” class which inherits directly from the “Entity” class however it seems that *time* and *room* are very similar classes. For consistency in design we suggest that in our Extensible Model these should inherit the properties of a *container* class (itself a resource) which is used to contain sets of other resources.

There are also 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, Fig. 3 shows all the constraints we modelled apart from the clash constraint which is implemented as a function in the Exam class. It would be nice if all the constraints were modelled in the same way as this would allow all existing constraints to be extended and for all constraints to be 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 we were creating was highly interactive it needed to be very responsive. The STTL interpreter proved to be too slow for our purposes and so the evaluation function was re-implemented in Java using the STTL simply as the data format for input and output.

From this experience we found that STTL was of most use as a data format for specifying instances and solutions precisely, whilst the evaluation functions and problem specifications were largely extraneous. It was found to be a relatively simple task to translate data from different formats into STTL.

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 model 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 languages.

Our goal is to create a small and simple subset of Classes which are required to model the examination timetabling problem, but that can be extended or added to model other timetabling problems. The model will be based on the structure of the problem domain and its solution 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 [6] summarised as:

- Generality
- Completeness of problem
- Ease of translation

This can be augmented with the additional requirement, *Ease of modelling*. These two properties provide an actual incentive for adopting this flexible model over other formats which exist. Ease of modelling suggests that this framework will actually make it easier to model timetabling problems than using a general language and is achieved in two ways:

1. Defined hierarchical framework
2. Reusable components

This framework will define model for the exam timetabling problem but can be extended to model other timetabling problems. It may well be that it won't 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 to use a subset of the well defined UML language to specify our framework and use the standard inheritance mechanism to create the flexibility we require. In the examples and terminology below the Java language is assumed but there is no reason that the design cannot be implemented in another language.

An ontology for constructing scheduling systems is proposed in [14]. 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 this approach is also taken in our model described below.

Based on all these ideas we propose an extensible model based on the constraint satisfaction problem built up in three layers:

1. Constraint Satisfaction Problem
2. General Timetabling Problem
3. University Examination Timetabling Problem

Each layer builds upon the previous layer adding problem specific resources and constraints. Once the lower layers have been implemented they can be re-used for different timetabling 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 ET problem.

5.1 The Constraint Satisfaction Problem layer:

The lowest level we consider is the constraint satisfaction problem, of which timetabling is an example. This problem simply consists of constraints that need to be satisfied, a ‘Resource’ class is added representing anything that is not a constraint.

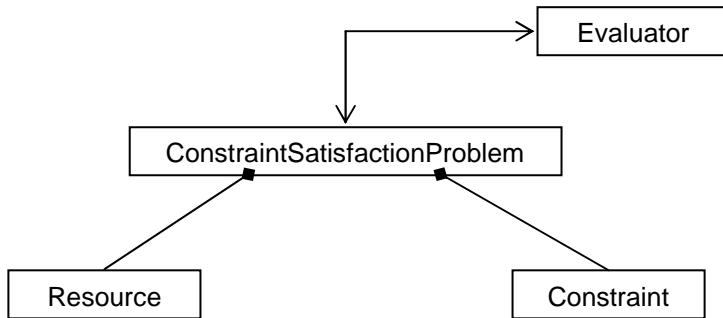


Fig. 4. The classes present in the Constraint Satisfaction Model

Constraints are modelled as functional classes. Each Constraint implements the methods shown in **Table 1**. The getViolationCount() method contains the logic for specifying the Constraint.

Table 1. Description of the Constraint class

Constraint Class	
getViolationCount()	Returns the number of violations of this Constraint found in the problem.
getWeight()	Returns the weight to be applied to violations of this constraint to calculate the cost of this solution.
isHard()	Returns true only if this is a hard constraint.

By storing attributes for the weight assigned to violations of this constraint and whether or not the constraint is *hard* or *soft* each Constraint class becomes responsible for evaluating itself. An overall evaluation function in an “Evaluator” class can then aggregate all these evaluations into the global evaluation function for the entire problem.

The final class introduced here is the Evaluator which is responsible for evaluating instances of this abstract Constraint Satisfaction problem.

Table 2. Description of the Evaluator Class

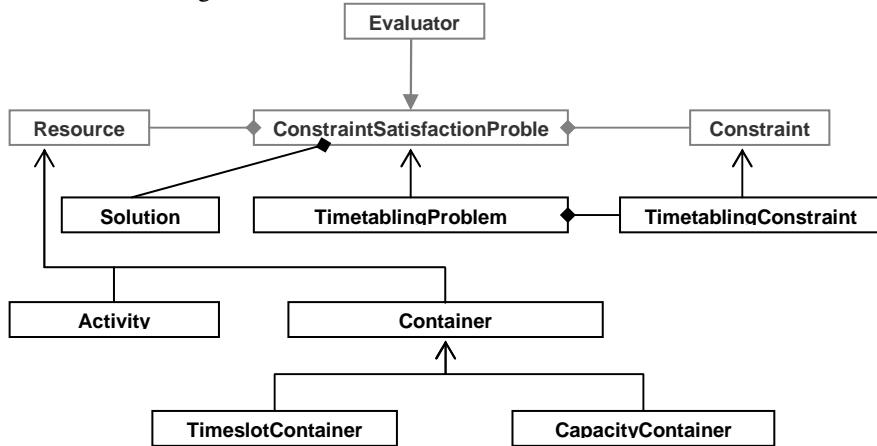
Evaluator Class	
Evaluate()	Calculates the cost of the current problem.
isFeasible()	Returns true only if no hard constraints have been violated.

In our instance, the actual evaluate ‘function’ is very simple, and can be implemented in few lines in Java:

```
public int evaluate(){
    int cost = 0;
    for (Constraint constraint:
        problem.getConstraints()) {
        cost += constraint.getViolationCount() *
            constraint.getWeight();
    }
    return cost;
}
```

5.2 The General Timetabling Problem

This model can then be extended for the abstract General Timetabling Problem, as illustrated below in figure 5:

**Fig. 5.** The Classes in the General Timetabling Model

The representation of time is one of the most difficult design decisions to make in a model such as this. As Time is not a Constraint we choose to model Time as a sequence of Timeslots, implemented using our Container interface to which Activities can be assigned. Each TimeslotContainer is specified with a duration and an order,

this simple representation could easily be extended with more information such as day/week information or whether a break exists beforehand.

The solution is represented by a completely new Solution class which stores the assignment of Activities to Containers.

Table 3. Description of the Classes found in the General Timetabling Problem

Class	Description
Activity	Any activity that is to be timetabled.
Solution	Stores the container each activity has been timetabled to
TimetablingConstraint	Constraints that can access the Timetabling resources
Container	A container where an activity can be timetabled
CapacityContainer	A container with a limit to the number of resources that can be added
TimeslotContainer	An ordered container with a specified duration

5.3 The University Exam Timetabling Problem layer

With the lower layers taken care of the ETP layer can be modelled relatively easily. Note that no work is needed to change the default Evaluator or Solution classes. In fact the only classes introduced here are those that directly map the abstract Timetabling problem to the real world Exam Timetabling application. It is envisioned that further timetabling problems can be modelled using this framework with similar ease.

The following classes and constraints are introduced to the model to implement the ETP:

Table 4. Resources in the Exam Timetabling Problem

Resource	Description
Exam	Models exam activities and their enrolments. Enrolments are lists of students taking this exam. As the activity resource is extended the name and duration attributes are already implemented.
Student	Models a student as a resource.
Room	A Container Class in which exam activities can be scheduled.

A working prototype of this model was built using Java and shown to work with our existing STTL data using a simple parser. As our design only specifies the interface of the model we were able to build in a number of optimizations to make our model

efficient at handling large data sets. The complete API specification for our model can be found online at:

<http://www.informatics.sussex.ac.uk/users/djr23/emdocs>

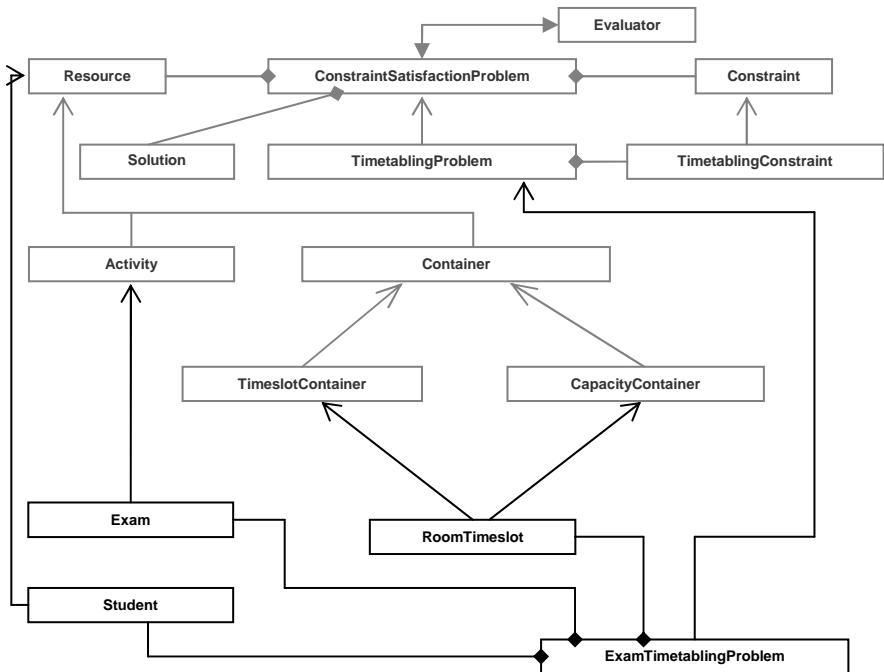


Fig. 6. Classes in the complete Exam Timetabling Problem model

Some of the Constraint classes register event listeners with the Solution class so they are notified of any changes to the Solution. This allows an incremental approach to counting the violations of each constraint and much improved performance.

6 Future Work

In this paper an attempt to design an “extensible” modelling framework, with the aim of simplifying the modelling process for many timetabling problems, is reported and applicability of this approach is demonstrated to model the Examination Timetabling problem. However, to demonstrate the extensibility of the modelling framework, it will be necessary to show that the model works for other timetabling applications and that the same design consistency can be applied across different problems in this domain. One possibility is to set up an online repository where these different applications of the model (documentation, implementations and problem data) can be accessed. We welcome any use of this model, especially in real world systems or applications to other timetabling problems.

An alternative to the use of STTL for data format for the Timetabling models is to store data as a simple XML document containing the information needed to instantiate each Class in the model. The logic and specification of the actual problem would remain in the implementation language but the instance data could be exchanged in this format, regardless of what language the model was implemented in. It would also be useful to create parsers for reading and saving to other data formats such as the Carter data format.

7 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 inline 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 exam timetabling model) 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. Fisher, J.G. and R.R. Shier, *A Heuristic Procedure for Large-Scale examination scheduling problems*. Congressus Numerantium, 1983. **39**: p. 399-409.
2. Burke, E.K., D. Elliman, P. Ford, and R. Weare. *Examination Timetabling in British Universities - A Survey*. in PATAT. 1995.
3. Gaspero, L.D. and A. Schaerf, *Tabu Search Techniques for Examination Timetabling* in *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III* 2001 Springer-Verlag. p. 104-117
4. The Timetabling Problem Database, 2003, retrieved on 30/01/2006 from <http://www.or.ms.unimelb.edu.au/timetabling.html>
5. Burke, E.K., J.H. Kingston, and P.A. Pepper, *A Standard Data Format for Timetabling Instances* in *Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II* 1998 Springer-Verlag. p. 213-222
6. Burke, E.K. and J.H. Kingston, *A Standard Format for Timetabling Instances*. Lecture Notes In Computer Science, 1997. **1408**.
7. TSPLIB-A library of travelling salesman and related problem instances, 1995, retrieved on January 2006 from <http://softlib.rice.edu/tsplib.html>
8. Kingston, J.H., *Modelling Timetabling Problems with STTL* in *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III* 2001 Springer-Verlag. p. 309-321
9. A user's guide to the STTL Timetabling Language, retrieved on January 2006 from <http://www.it.usyd.edu.au/~jeff/tsttl1.ps>

10. Ozcan, E., *Towards an XML based standard for Timetabling Problems: TTML*, in *Multidisciplinary Scheduling: Theory and Applications: 1st International Conference, Mista '03 Nottingham, UK, 13-15 August 2003. Selected Papers*, G. Kendall, et al., Editors. 2005, Springer-Verlag.
11. Reis, L.s.P. and E. Oliveira, *A Language for Specifying Complete Timetabling Problems*, in *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*. 2001, Springer-Verlag. p. 322-341.
12. Ranson, D., *Interactive Visualisations for the Generation, Evaluation and Analysis of heuristics in scheduling: Thesis Progress Report 2004*. 2004, Progress Report, University of Sussex.
13. Ranson, D. and P.C.-H. Cheng. *Graphical Tools for Heuristic Visualization*. in *Multidisciplinary International Conference on Scheduling: Theory and Applications*. 2005. New York, USA.
14. Smith, S. and M. Becker, *An Ontology for Constructing Scheduling Systems*, in *Working Notes of 1997 AAAI Symposium on Ontological Engineering*. 1997, AAAI Press.

Generating Personnel Schedules in an Industrial Setting Using a Tabu Search Algorithm

Pascal Tellier¹ and George White²

¹ PrairieFyre Software Inc.,

555 Legget Dr., Kanata K2K 2X3, Canada

pascal@prairiefyre.com

² School of Information Technology and Engineering,

University of Ottawa, Ottawa K1N 6N5, Canada

white@site.uottawa.ca

Abstract. We describe a system designed to be used in an industrial setting for the scheduling of employees in a Contact Centre. The demand for the employees' services is driven by an estimated forecast for each period of operation, currently set with a duration of 15 minutes. The employees are drawn preferentially from a homogeneous pool. The constraints are set by the conditions of employment and must satisfy the requirements both of the company and the desires of individual employees who may have very diverse interests. The system uses an initial scheduling module that constructs a feasible schedule followed by an optimizing tabu search based heuristic optimizer to cast the final schedule. This system is evaluated both with artificially constructed data and real industrial data.

1 Introduction

One of the most important problems that arises in organizations composed of more than a few workers is the management of personnel. Issues that arise in this arena can be very complex and failure to observe the rules of play can result in a wide range of outcomes ranging from isolated grumbling, reduction in general staff morale, work stoppages, strikes, mass resignations and expensive lawsuits. The financial impact of these factors can be enormous.

The issues involved in this area are such things as:

- who does what jobs
- when do they do it
- where do they do it
- who do they do it with
- how much are they paid

These issues are formalized by the constraints whose origins lay in the customs of the local workplace culture, collective agreements, government legislation, and informal arrangements between the persons responsible for the scheduling of personnel and resources and those directly involved by the schedule. Errors, misunderstanding and misinterpretation of the schedules can raise a host of problems.

A complicating factor is the emergence of a class of so-called “temporary workers” who work only as required. The constraints involved in the scheduling of these people can be more complicated to formulate and more difficult to respect because of the various start and stop times that can be different for each person and may depend on the day of the week, the month of the year, forecasted demand and/or a spectrum of other considerations specific to the industry. All this must be done within the framework of corporate objectives. The overall goal is to balance meeting the service level while optimizing the budget and respecting employees’ rights and preferences to keep them happy.

2 Tour Scheduling in Contact Centres

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 [1] has named this type of labour scheduling *tour scheduling*. If the work force can be classified into different types (usually corresponding to different skill levels) it is described as *heterogeneous*. If we are dealing with one skill level, as is the case with this paper, it is called *homogeneous*.

Recent reviews of the tour scheduling literature have been published by Alfares [2] and by Ernst *et al*[3]. 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. Alfares [2] has found 14 different criteria used by various authors to formulate these objective functions. Of these criteria, the ones germane to the present problem are the total daily and weekly hours worked, under and over staffing and employee satisfaction.

Alfares has also partitioned the methods used for optimizing functions that incorporate these criteria into 10 categories. The work reported here uses the tabu search (TS) created by Glover [4].

As stated by Ernst *et al.*: [3]

.... heuristics are generally the method of choice for rostering software designed to deal with messy real world objectives and constraints that do not solve easily with a mathematical programming formulation.

Contact centres are an integral part of many businesses. They are composed of their staff whose job it is to answer or initiate telephone calls, to answer questions or to solicit business. The literature on contact centres has been reviewed by Gans *et al.* [5]. Scheduling the staff in these centres is almost never solved to optimality due to the complexity of the requirements. Suboptimal solutions that can be used in practice are formulated using heuristic methods.

This paper describes a typical problem that arises in the scheduling of personnel in contact centres and describes a solution based on tabu search. The system, part of which is described here, was undertaken by PrairieFyre Software Inc.

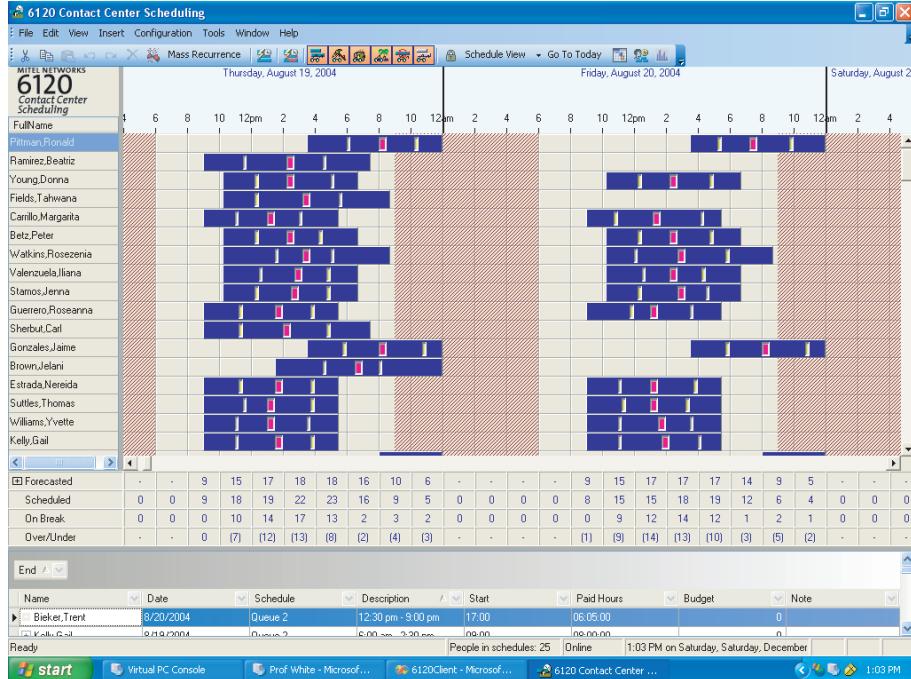


Fig. 1. Typical schedule

3 Problem Specifics

The problem at hand is to generate a schedule $s_{i,j}$ where: i is an integer index specifying a *person* and j is an integer index specifying a *period*, a fixed interval of time. $s_{i,j}$ represents the *state* of *person_i* during *period_j*. This *state* can have values corresponding to: on-the-job, not working, on paid break, on unpaid break, on vacation, and a few other things. For our purposes we can simplify the value of $s_{i,j}$ such that it equals 1 when *person_i* is on-the-job during *period_j* and 0 otherwise. Then the number of persons working during *period_i* is given by:

$$w_j = \sum_i s_{i,j}$$

The total number of person-periods working over the entire schedule is given by:

$$W = \sum_j w_j$$

For each *period_j* a forecast f_j is obtained by some method such as reference to historical data or by a modified Erlang calculation. This forecast is an estimate of the number of persons “on duty” during the period concerned. This is just equal to the number of persons at work at the time minus the number having a

break. If $w_j - f_j$ is negative, there are too few persons working and the schedule is said to be underscheduled during that period. If the difference is positive, there are too many persons working during that period and the schedule is said to be overscheduled during that period. The *penalty* assessed for the entire schedule S is calculated as

$$P(S) = \sum_j (w_j - f_j)^2$$

In the field it may be desirable to change the form of the penalty to reflect the overall number of people being scheduled. A shortfall of 2 persons is not too important when 50 people are supposed to be scheduled, however a shortfall of 2 when 3 are to be scheduled can be very bad. A “typical” schedule looks like the one shown in figure 1.

Each person in the workforce and each shift have several important attributes. These are:

- Persons
 - name
 - scheduling priority
 - minimum hours between shifts
 - minimum daily hours
 - maximum daily hours
 - minimum weekly hours
 - maximum weekly hours
 - earliest start time (for each day of the week)
 - latest end time (for each day of the week)
- Shifts
 - name
 - type
 - typical hours
 - maximum hours
 - minimum hours
 - earliest start time
 - latest start time
- Breaks
 - name
 - duration
 - paid or not
 - shift length required to qualify
 - earliest start time after beginning of shift
 - latest start time after beginning of shift
 - minimum minutes before end of shift

The *persons* are the staff members to be scheduled. Each line on figure 1 is a graphic representation of the working day of a different person. The *shifts* are the periods during which a person is on the job. Each shift has zero or more *breaks* during which that person is off duty. A break may be paid or unpaid.

4 Constraints

A *feasible* schedule is one that satisfies a set of constraints. The constraint set used here can be divided into four parts.

4.1 Weekly Constraints

For all workers

- sum of working paid hours each week \leq maximum weekly hours for that worker
- sum of working paid hours each week \geq minimum weekly hours for that worker
- for all days, each person-shift must be separated by at least the minimum hours between shifts

4.2 Daily constraints

Each person-shift

- must start \geq earliest start time for that day for that person
- must end \leq latest end time for that day for that person
- must start \geq earliest start time for that shift
- must end \leq latest end time for that shift
- shift length \leq maximum daily hours for that person
- shift length \geq minimum daily hours for that person (if working that day)
- shift length \leq maximum shift length
- shift length \geq minimum shift length

4.3 Breaks

Each shift-break

- length of shift into which break is embedded \geq shift length required to qualify
- start time \geq earliest start time after beginning of shift
- start time \leq latest start time after beginning of shift
- start time \leq shift end time - minimum hours before end of shift

4.4 Global constraints

Since this work is designed for use in a real contact centre a number of practical features have been built in. These features generally remove some of the constraints, thereby reducing the penalty of an eventual solution. Some of these are

- override the available times specified by the employees
- override the days specified as unavailable by the employees
- calculate penalty only during “office hours”

There are six classes of overrides that can be used by the scheduling officers.

4.5 Other

Another feature completely inhibits optimization of one or more days of the schedule. Use of this feature will worsen the quality of the optimization but allows the unoptimized days to be scheduled “as is” *i.e.* exactly as cast by an existing initial schedule generator or cast manually or partially manually by the scheduling officer. Not all of the constraints have been implemented in the current version.

5 Goal

Each day of interest is divided into a number of periods of the same duration. The results described here refer to the case where the day consists of 96 periods, each one lasting 15 minutes. Each period is assigned a forecast of the number of workers required during that period. The number who are actually on duty should ideally be equal to the number forecast for that period. Each period is assigned a penalty equal to the square of the difference between the forecasted value and the number actually on duty. The penalty corresponding to a complete day is defined to be the sum of the penalties of all its periods. The goal of the scheduler is to cast a schedule such that all constraints are satisfied and the penalty of each day is as low as possible.

The forecasted number of employees required for each period is calculated by using a modified Erlang calculation. These numbers may be modified by a supervisor to account for local or unusual circumstances.

In practice, it may be preferable to attempt to minimize a penalty function that takes into account the total number of employees being scheduled and minimize the percentage deviation from the desired number of employees rather than the square of the absolute deviation. At present, however, we are concerned exclusively with absolute deviations.

With this explanation we can write formally that the problem consists in casting S such that:

1. $P(S)$ is minimized
2. the constraint set $c_i \odot a_i$ is satisfied
 - c_i is the left hand side of the constraint
 - \odot is a relational operator
 - a_i is the constant on the right hand side

The implementation has been constructed in such a way that other possible goals can be easily incorporated. There are two alternate goals under consideration:

1. The number of *understaffed* periods is zero and the overall penalty as previously described is minimized.
2. The total dollar cost (salary) of the workers (including various rates of pay and overtime) is minimized while supplying a minimum level of service.

6 Tabu Search

The well known tabu search methodology [4] has proven to be a robust method of finding heuristic minimums of complex scheduling problems in reasonable times [6] [7] [8] [9]. Tabu search (TS) is a heuristic procedure designed to guide the search for an optimal schedule through the trap of local minima. Starting from an initial solution s_0 the TS algorithm iteratively explores a subset V^* of the neighbourhood $N(s)$ of a current solution s . The member of V^* that gives the minimum value of the objective function becomes the new current solution independently of whether its value is better or worse than the value corresponding to s . If $N(s)$ is not too large, it is possible and convenient to take $V^* = N(s)$. This strategy may induce cycling.

In order to prevent cycling, the algorithm calculates a so-called tabu list, a list of moves which the search portion of the procedure is forbidden to make. This is a list of the last k accepted moves (in reverse order) and it is often implemented as a queue of fixed size. At equilibrium, when a new move is added, the oldest one is discarded.

There is also a mechanism that may override the tabu status of a move. If a move could give a large improvement (reduction) of the objective function, then its tabu status is dropped and the resulting solution is accepted as the new current one. This is implemented by defining an aspiration function A that, for each value v of the objective function, returns another value v' that represents the value that the algorithm aspires to reach from v . Given a current solution s , the objective function $f(s)$, and a neighbour solution s' , then if $f(s') \leq A(f(s))$ then s' can be accepted as the new current solution, even if s' is in the tabu list.

The procedure stops either after a given number of iterations without improvements or when the value of the objective function in the current solution reaches a given lower bound.

The main control parameters of the procedure are the length of the tabu list, the aspiration function A , the cardinality of the set V^* of neighbour solutions tested at each iteration, and $Tsmax$, the maximum number of iterations allowed that do not improve the objective function.

The tabu search strategy is implemented in the following way:

1. An initial schedule $s^0 \in X$ is chosen by an approximate but rapid method. X is a set of *feasible* solutions.
2. Certain variables controlling the stopping conditions are initialized here. These variables consist of the maximum number of iterations permitted, the maximum time the program is permitted to run and other similar quantities. These variables are used to determine whether the program should terminate or continue at step 4. The tabu list T is implemented as a linked list. The tenure was given a fixed value of 10. The value of the objective function corresponding to the initial solution is calculated.
3. The aspiration function is set to return the minimum value found so far.
4. At this point the program enters its main loop which exits when one of the following conditions become true:

- the best solution found yet has a penalty = 0.
 - the number of iterations without improvement \geq some maximum value, currently set to 1,000.
5. Each schedule $s \in X$ can be modified by applying a simple perturbation called a *move* to s . The neighbourhood, $N(s)$, consists of all feasible solutions that can be obtained by applying the perturbation to s . There are two separate move classes considered:
- swapping two periods of one person
 - extending or shortening the schedule of one person by one period at the beginning or the end of their shift. This has the effect of either lengthening or shortening the shift or sliding it forwards or backwards by a small amount.
- For each person, these are applied sequentially. First all swapping possibilities are considered. Next, depending on whether the schedule is over or under scheduled, shortening or lengthening the shift is tried. Finally sliding the shift forward and backward is considered.
6. A subset of feasible solutions $V^* \subseteq N(s)$ is constructed from the elements of $N(s)$ in the following way. An element $s' \in N(s)$ is placed in V^* unless s' is in the tabu list T . However, even if $s' \in T$, it will be still be placed in V^* if $f(s') < A(f(s))$. $f(s')$ is the penalty or objective function associated with schedule s' and $A(f(s))$ is the value of the aspiration function associated with each value of the objective function f . Thus $A(f(s))$ is the value of the aspiration function associated with schedule s . In our work, $A(f(s)) = f(s^0)$ where s^0 is the best schedule found so far *i.e.* $f(s^0)$ is the lowest value of the objective function calculated as of yet. Note that s is not a member of its own neighbourhood. Therefore $s \notin V^*$.
7. The penalties $f(s^*)$ associated with each $s^* \in V^*$ are calculated and the value s^* having the lowest value $f(s^*)$ is chosen. This is done even if s^* is worse than the current schedule. There are cases where the shift and break requirements are so strict that no legal moves are possible. In this case, the step is exited immediately.
8. The tabu list is updated with the new solution found s^* . Rather than storing the entire solution in the list, the *move itself* is stored, making list management easier. Any list entries whose tenure ≥ 10 are discarded.
9. The new solution is compared to the best solution found so far s^0 .
10. If the new solution is better, it replaces the now former best solution.
11. The current solution becomes the new solution.

7 The initial solution

The initial solution s_0 is constructed by an existing algorithm. The quality of this solution varies widely from instance to instance, sometimes producing good solutions and sometimes producing bad ones. The present program starts by calling a module we call the initial solution evaluator that analyzes the quality of this solution and may alter it severely, removing some shift instances completely if the solution is overscheduled and adding new shift instances if it is underscheduled.

8 Evaluation of the algorithm

The algorithm has been tested both with contrived data (test data) *i.e.* data that has been constructed to test specific features of the specification and the algorithm, and customer data *i.e.* real data furnished by existing customers. In both cases measurements have been made of the algorithm's ability to cast heuristically optimal schedules. These tests have been made using the module that starts by generating initial schedules and without using this module. In the latter case, the tabu search starts from an empty schedule and uses its own initial schedule evaluator to build a crude initial schedule and then uses the tabu portion to optimize it. The results are shown in the table below.

Table 1. Some test results

Name	length	conditional	before	after	reduction	% reduction	time (sec)
Test1	1 day	with initial	128	13	115	0.900	35.9
Test2	1 day	with	983	61	922	0.938	36.8
		without	5095	71	5024	0.986	33.0
Test3	7 days	with	1742	145	1597	0.917	218.0
Cust1	1 day	with	6241	32	6209	0.995	90.2
		without	1381	24	1357	0.983	13.6

The relative reduction in these examples is never less than 0.90, showing that the tabu algorithm is capable of reducing the penalty of a schedule to less than 10% of its initial value. The best case shown was obtained for customer data (from a real customer!) and reduced the penalty from 6241 to 32, or about one half of one percent of its original value.

Obviously the initial schedule has a heavy influence on the penalty of the final schedule. The relative reduction values that look so impressive are partially due to the bad results sometimes produced when the penalty of the initial schedule is calculated.

In all cases the TS algorithm was able to reduce the penalties of schedules significantly (better than 90%).

9 Implementation Details

The algorithm was coded in C# using the Microsoft Visual Studio .NET 2003 IDE. The results quoted in the table above were measured using a single processor Pentium 4 System with 1 GByte of RAM clocked at 2.40 GHz.

10 Conclusions

After testing with both artificial and real data the heuristic optimizer based on the classic tabu search paradigm was proven to generate employee schedules that

satisfy the constraints and have penalties much lower than schedules generated by the initial construction algorithm. Typically the penalty associated with an initial schedule is reduced by 90% or more.

A visual inspection of the final schedule often gives the impression that it is optimal, although one can never be sure. There does not appear to be a test suite for personnel problems such as the one available for examination schedules, so there is no way of comparing our results with those found using other systems.

References

1. K. R. Baker. Workforce allocation in cyclic scheduling problems: A survey. *Operation Research Quarterly*, 27:155–167, 1976.
2. H. K. Alfares. Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research*, 127:145–175, 2004.
3. A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153:3–27, 2004.
4. F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.
5. N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: Tutorial, review and research prospects. *Manufacturing and Service Operations Management*, 5:79–141, 2003.
6. F. Glover and C. McMillan. The general employee scheduling problem: An integration of management science and artificial intelligence. *Computers and Operations Research*, 13:563–593, 1986.
7. G.M. White, B.S. Xie, and S. Zonjic. Using tabu search with longer-term memory and relaxation to create examination schedules. *European Journal of Operational Research*, 153:80–91, 2004.
8. E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.
9. Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetables. *Lecture Notes in Computer Science*, 2079:104–117, 2001.

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 Linear Linkage Encoding 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 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 [18].

This paper is organized as follows: We first define the grouping problems and common representations for them. The fundamentals of Linear Linkage Encoding 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

$$V = V_1 \cup V_2 \cup V_3 \dots \cup V_N \text{ and } V_i \cap V_j = \emptyset \text{ where } i \neq j.$$

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 bin packing problem, 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 [24].

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 disjointed oriented pseudo path graphs G_1, G_2, \dots, G_m with the following properties:

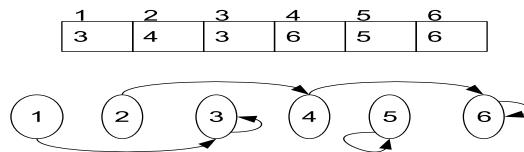


Fig. 1. LLE Array and LOP Graphs

- 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
 - a. $|E(G)| = |V(G)|$
 - b. each G_i has only one ending node with an *in-degree* of 2 and *out-degree* of 1.
 - c. 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.

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 [20]. Paquete et. al. [23] 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. [22] 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

Graph Coloring (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 Graph Coloring Problem 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 as in DSATUR. 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 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 proceeded 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 present 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 cooperating through crossover and mutation. Tight lower and upper bounds can be found based on the maximal clique and maximal degree of the graph. 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 Niched Pareto 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 mates. 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 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 Manhattan distance in terms of color and conflict values in the individuals. The objective functions c_{ix} and c_{jx} represents 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 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 [25], 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

Linear linkage encoding 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 destructing these building blocks. However, for small problem instances, one-point crossover in LLE is reported to generate satisfactory results for 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 Graph Coloring Problem 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.

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 Greedy Partition Crossover 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 presented 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 largest set in parent 2 (1) is transmitted to the child. Finally (2) is assigned as the last group. After sorting according to 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.

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 lowest index number (GPX-LI).
14: else
15:   sort group ids according to cardinality (GPX-CB).
16: end if
```

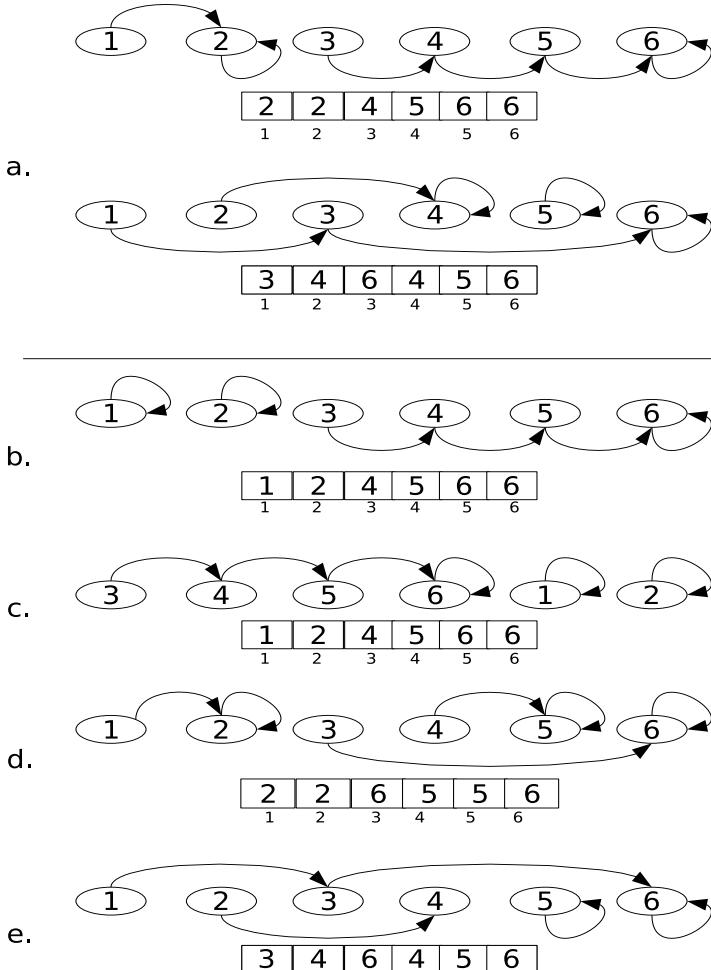


Fig. 2. a) Two Parents in LLE Array and LOP Graph form. b) Resulting offspring from Greedy Partition Crossover - Lowest Index Ordering c) Resulting offspring from Greedy Partition Crossover - Cardinality Based Ordering. d) Resulting offspring from Lowest Index First Crossover. e) Resulting offspring from Lowest Index Max Crossover.

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:

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 presented in Algorithm 2.

The application of LIFX on the parents in Figure 2 would be as follows: Assuming we begin with 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.

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

```

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 amalgamate of LIFX and GPX. 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 presented 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: $C_1 = (1, 3, 6)$, $C_2 = (2, 4)$, $C_3 = (5)$

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

```

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.

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, λ (non elitist) + μ (elitist individuals, one for each number of colors within the searching range) individuals produce λ 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 forms the next generation.

4 Experiments

In our tests, we use several graphs from the DIMACS Challenge Suite [18]. The general test setup is summarized in Table 1.

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

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	?

In Table 2, we present 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.

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

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 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.

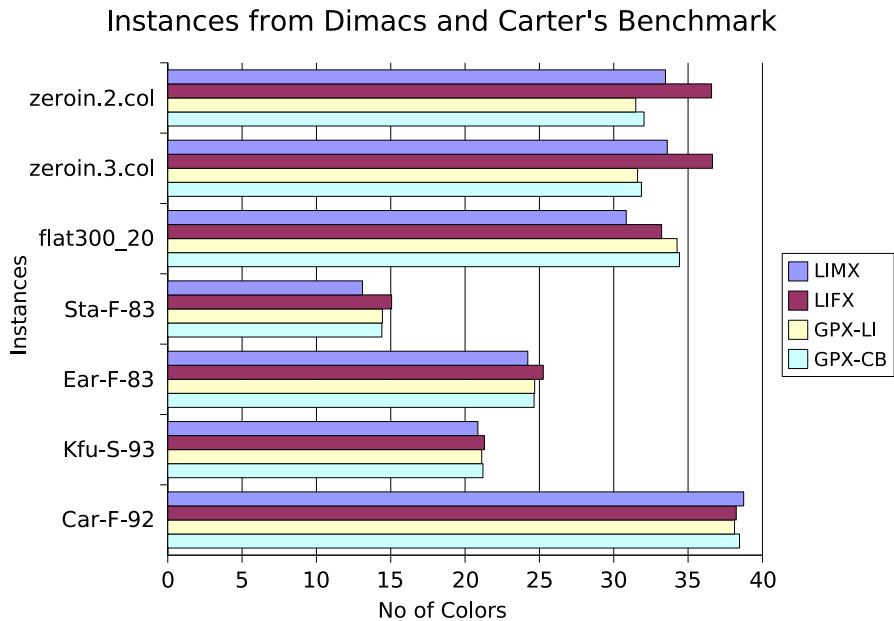


Fig. 3. Average number of colors (groups) for some instances in DIMACS and Carter's Benchmark.

We have also presented graph coloring algorithm results of Kirovski *et. al.* [17] for two set of parameters (Kirovski B and Kirovski C). Kirovski'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, post processing lottery-scheduling iterative improvement. With respect to Kirovski's solutions, our crossovers gave similar and for some instances better results however when the instance becomes larger and more difficult, Kirovski'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 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 4 presents some instances taken from the Carter’s Benchmark [5]. We again present 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 presented.

Table 5. Best colorings obtained for the instances in the Carter’s 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

For 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 [21]). Like DIMACS instances, the performance of the graphs with vertices above 500 suffered due to the limit on the maximum number of generations. For instances, our crossovers gave similar results in terms of best grouping obtained. Generally they obtained colorings equal or one color behind colorings of Carter et. al and Caramia et. al, and better than 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 presented. 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. Linear Linkage Encoding 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.

6 Acknowledgements

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

References

1. Avanthay C., Hertz A. and Zufferey N. Variable neighborhood search for graph coloring, European Journal of Operational Research (2003) 151 pp 379-388.
2. Brelaz, D. "New Methods to Color Vertices of a Graph", Communications of the ACM (1979) 22 pp 251-256.
3. Burke, E.K., Newall, J. and Weare, R.F. "A Memetic Algorithm for University Exam Timetabling". In: Burke, E.; Ross, P. (eds.): Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August/September 1995. Selected Papers. Lecture Notes in Computer Science 1153, (1996) Springer-Verlag, Berlin Heidelberg New York 241–250.
4. Caramia, M., Dell'Olmo, P., and Italiano, G.F. "New algorithms for examination timetabling". In: Nher, S.; Wagner, D., (eds.): Algorithm Engineering 4th International Workshop, WAE 2000, Saarbrücken, Germany, September 2000. Proceedings. Lecture Notes in Computer Science 1982, Springer-Verlag, Berlin Heidelberg New York, (2001), 230-241.
5. Carter, M. W, Laporte, G. and Lee, S.T. "Examination timetabling: algorithmic strategies and applications.", Journal of the Operational Research Society, (1996) 47:373-383.
6. Du, J., Korkmaz E., Alhajj R., Barker K., "Novel Clustering Approach that Employs Genetic Algorithm with New Representation Scheme and Multiple Objectives." In Proceedings of 6th International Conference on Data Warehousing and Knowledge Discovery - DaWaK (2004), Zaragoza, Spain.

7. Even, S., Itai, A. and Shamir, A. "On the Complexity of Timetable and Multicommodity Flow Problems", SIAM J. Comput., 5(4):691-703 (1976).
8. Falkenauer, E. "Genetic Algorithms and Grouping Problems", John Wiley & Sons (1998).
9. Fleurent, C. and Ferland, J.A. "Genetic and Hybrid Algorithms for Graph Coloring", Annals of Operations Research 63 pp 437-461 (1996).
10. Galinier, P. and Hao, J.K., "Hybrid Evolutionary Algorithms for Graph Coloring", Journal of Combinatorial Optimization 3 pp 379-397 (1999).
11. Garey, M.R., Johnson, D.S., "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freedman San Francisco (1979).
12. Goldberg, D.E., "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison Wesley, Reading, Massachusetts (1989).
13. Holland, J. "Adaptation in Natural and Artificial Systems", University of Michigan Press (1975).
14. Hertz, A., and De Werra, D. "Using Tabu Search Techniques for Graph Coloring", Computing, (1987) 39 pp 345-351.
15. Horn, J., Nafpliotis, N., Goldberg, D. E., "A Niched Pareto Genetic Algorithm for Multi-objective Optimization", Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, (1994) vol 1. pp 82-87, Piscataway, New Jersey.
16. Johnson, D.S, Aragon, C.R, McGeoch, L.A, and Schevon, C., "Optimization by Simulated Annealing: An Experimental Evaluation: Part II, Graph Coloring and Number Partitioning", Operations Research, (1991) 39 (3) pp 378-406.
17. Kirovski D., and Potkonjak. M., Efficient Coloring of a Large Spectrum of Graphs. 35th Design Automation Conference Proceedings, (1998) pp. 427-432.
18. Johnson, D.S, Trick, M.A, (Editors), "Cliques, Coloring and Satisfiability", DIMACS Series in Discrete Mathematics and Theoretical Computer Science", (1996) Vol. 26, American Mathematical Society.
19. Leighton, F.T, "A Graph Coloring Algorithm for large Scheduling Problems", Journal of Research of the National Bureau Standard, (1979) 84 pp 79-100.
20. Terashima-Marn, H., Ross, P. and Valenzuela-Rendn, M. "Clique-Based Crossover for Solving the Timetabling Problem with Gas", Proc. of the Congress on Evolutionary Computation, (1999) pp. 1200-1206.
21. Merlot, L.T.G., Boland, N., Hughes B.D., and Stuckey, P.J. "A Hybrid Algorithm for the Examination Timetabling Problem." In: Burke, E.; De Causmaecker, P. (eds.): Proceedings of Practice and Theory of Automated Timetabling, Fourth International Conference, Gent, Belgium, (2002) pp 348-371.
22. Ozcan, E., Ersoy, E., "Final Exam Scheduler - FES", Proc. of 2005 IEEE Congress on Evolutionary Computation, (2005) vol. 2, pp 1356-1363.
23. Paquete, L. F. and Fonseca C. M. "A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms", Proc. of the 4th Metaheuristics International Conference (MIC 2001), (2001) pp. 149-154, Porto.
24. Radcliffe, N.J, "Formal Analysis and random respectful recombination", In Proceedings of the 4th International Conference on Genetic Algorithm, (1991) pp. 222-229
25. Ramani, A., Aloul F.A, Markov, I and Sakallah, K.A., "Breaking Instance-Independent Symmetries in Exact Graph Coloring." Design Automation and Test Conference in Europe, (2004) pp 324-329
26. Wong, T., Cote, P. and Gely, P. "Final exam timetabling: a practical approach", Canadian Conference on Electrical and Computer Engineering, Winnipeg, CA, (2002) vol.2, pp. 726-731.

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

Christine A. White¹, Emilina Nano², Diem-Hang Nguyen-Ngoc², and George M. White²

¹ Dept. of Medicine, Div. of Nephrology,
Queen's University, Kingston K7L 2V6, Canada
cw38@post.queensu.ca

² School of Information Technology and Engineering,
University of Ottawa,
Ottawa K1N 6N5, Canada
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

In work described in a previous PATAT conference [1] 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 [2] 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 are 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. The constraints are examined in detail and the penalties associated with them are discussed in the earlier paper [1]. An example of a call schedule is shown in figure 1.

		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

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 penalties 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 trainees 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 which are ones that can be evaluated simply by scanning each line separately.

Vertical penalties are those that arise from consecutive days. The first example of two nights referred to above 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. 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 higher value. Table 1 lists the various defects that each night’s shift might have and the corresponding penalties.

Table 1. Conditions and their penalties

Condition	Penalty
one student missing	5
student replaces missing junior - senior is on student’s team	10
two student 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
anything else	500

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. For real data, the results all exhibited the same overall behaviour. We believe this is so because the staff mix and numbers were chosen by a supervisor having much experience casting schedules by hand, thus they all exhibit a certain

homogeneity. This paper summarizes the four algorithms used and discusses the results obtained from each one.

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 analyzed 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 (TS) whose entries in the single tabu list have a fixed tenure was the original algorithm implemented in the project [1]. This algorithm was rerun with actual hospital data keeping the tenure fixed but varying its value in different runs. This algorithm is deterministic. A run always returns the same schedule having the same penalty. 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

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]. 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.

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
std dev	82.8	0.6

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 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. As above, this method is not deterministic. The results are summarized in table 4.

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
std dev	353.9	0.6

2.4 IDWalk

This method, called Intensification/Diversification Walk (or IDWalk), was introduced by B. Neveu *et al.* [5] 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 this 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. Otherwise, $SpareNeighbour$ was set to "any" and any one of the rejected neighbours is chosen randomly for the next move.

In this investigation, $S = 1000$ and $Max = 378$. The two choices for $SpareNeigh$ were tried and it was found that the value, "best" gave the 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

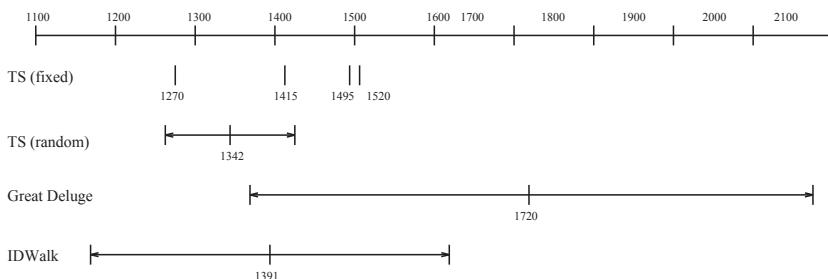
	penalty	execution time (s.)
min	1160	57.4
max	1810	77.5
mean	1394	68.1
std dev	225.1	7.7

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

**Fig. 2.** Comparison of results

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 on 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 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 on figure 2.

The IDWalk method yielded a mean penalty of 1394, 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 worst 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 on 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 result in distributions having identical means, at the 95% level. When TS (random) and Great Deluge are compared the means were found to be 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

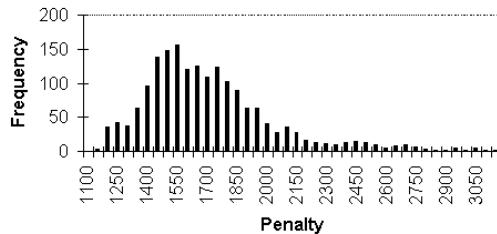


Fig. 3. Histogram of 1800 penalties obtained for the Great Deluge algorithm

once). 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 for the deterministic ones.

However, if execution time is not critical, the method of choice may be the Great Deluge which yields results rapidly, even if most of them are not very good. One of them may be very good indeed.

If it is desired to find the *minimum minimorum* and execution time is not a concern then IDWalk may well give the best results. If time is a concern, then it may be useful to run the Great Deluge multiple times and select the schedule having the smallest penalty. 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 an 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 penalty found in the comparative samples was obtained by IDWalk, simultaneously taking the longest time to get it. The lowest 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.

References

1. Christine A. White and George M. White. Scheduling doctors for clinical training unit rounds using tabu optimization. *PATAT IV: Lecture Notes in Computer Science*, 2740:120–128, 2003.
2. F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.
3. Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. *PATAT IV: Lecture Notes in Computer Science*, 2740:262–275, 2003.
4. Gunter Dueck. New optimization algorithms. *J. Comp. Phys.*, 104:86–92, 1993.
5. Bertrand Neveu, Gilles Trombettoni, and Fred Glover. Idwalk : A candidate list strategy with a simple diversification device. *CP 2004: Lecture Notes in Computer Science*, 3258:423–437, 2004.

Extended Abstracts

Tackling the university course timetabling problem with an aggregation approach

Mieke Adriaen, Patrick De Causmaecker[†], Peter Demeester
and Greet Vanden Berghe

KaHo Sint-Lieven
Information Technology
Gebroeders Desmetstraat 1, 9000 Gent, Belgium
{Mieke.Adriaen, Peter.Demeester, Greet.VandenBerghe}@kahosl.be

[†] Katholieke Universiteit Leuven Campus Kortrijk
Computer Science and Information Technology
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
Patrick.DeCausmaecker@kuleuven-kortrijk.be

1 Introduction

In this abstract we describe the university timetabling problem as it is perceived and solved at the KaHo Sint-Lieven School of Engineering. Timetabling is carried out manually by dragging and dropping events in a room-timeslot matrix, but potential conflicts are automatically spotted by the conflict module built into the application. We are now in the process of automatizing the construction of timetables itself. In the next sections we describe the specific timetabling problem at KaHo Sint-Lieven and the steps we follow to tackle it. The approach is based on a tabu search algorithm. Inspired by Kingston [13], we group sessions in order to make the timetabling problem less complex.

2 University Course Timetabling

University timetabling is probably one of the best studied timetabling problems [2,4,6,14] in academia. Bardadym [2] classifies university timetabling into 5 groups.

- faculty timetabling: assign qualified teachers to courses,
- class-teacher timetabling: assign courses with the smallest timetabling unit being a class of students,
- course scheduling: assign courses with the smallest scheduling unit being an individual student,
- examination scheduling: assign examinations to students such that students do not have two examinations at the same moment,
- classroom assignment: after assigning classes to teachers, assign these class-teacher couples to classrooms.

The problem considered in this abstract can be classified as course scheduling. That problem consists of placing events (which we will call sessions) in appropriate class rooms and timeslots, taking into account that the number of students attending a session has to fit into the room and that the duration of the session cannot be greater than the length of the timeslot it is assigned to. Other constraints that have to be taken into consideration are that students and lecturers cannot be at two places in the same timeslot and that a session can only be assigned to one timeslot and one room. The university timetabling problem that is studied in this paper is in general more complex than the secondary school timetabling problem. In the latter, timeslots are equal in length and the schedule is weekly repeated during a semester. At the KaHo Sint-Lieven School of Engineering, a timetable is typically constructed for a period of a semester (which is twelve weeks long). The problem however is that some subjects are taught every week; some sessions are taught every fortnight, others only the first six weeks of the semester, others only nine times a semester, ... It is even more complex since the timeslots are not equal in length, and since they can overlap. This has to be taken into consideration when constructing the timetable. Multiple lecturers can be assigned to lab sessions which are taken by large student groups. This makes the manual construction of timetables a hard and time consuming task. It usually takes a couple of days to remove all the conflicts in the timetable that appear once the semester has started.

The considered problem size for one semester is:

- 12 weeks,
- 133 different teacher groups,
- 212 different student groups,
- there are 100 class rooms of different sizes available,
- there are 20 different (sometimes overlapping) possible timeslots per day,
- 1215 sessions need to be scheduled.

Most university timetabling applications described in the literature have been developed by universities that experienced the need for automatizing their own timetabling problem [6]. Bardadym [2] remarks that most of these systems are usually very problem specific and can only be applied in that particular university. We are attempting to overcome that by developing a general timetabling framework [7].

3 Tabu Search Approach

The literature about university course timetabling teaches us that researchers applied different approaches to tackle the problem (see [14] for an overview). Meta-heuristics approaches are known to produce good results. Early papers (beginning 1990's) of Hertz [11,12] report good results applying tabu search [9] on the university course timetabling problem. More recent papers on university course timetabling combine tabu search with other search methods: Schaerf [15] applies a combination of randomized hill-climbing with tabu search, while Burke

et al. [5] use a hyperheuristic in which a tabu search algorithm is employed to select the lower level heuristics.

3.1 General Description of the OpenTS Framework

In this contribution we report on the OpenTS[10] framework and its employability on university course timetabling. That framework contains a Java based implementation of a tabu search algorithm. Developers who start applying the framework are expected to implement the problem specific interfaces. They thus describe how a solution will be represented, what will be the different neighborhoods, and how the different constraints will be evaluated. The OpenTS framework also offers the opportunity to dynamically adapt the length of the tabu list (reactive tabu list) and to switch between different neighborhoods.

3.2 Application of OpenTS to the Course Timetabling Problem

In this section we describe how we applied the OpenTS framework to the university course timetabling problem. We implemented the problem specific interfaces and incorporated the possibility to dynamically adapt the tabu length size and switch between neighborhoods.

- The implementation of the OpenTS Solution interface consists in this case of a two dimensional matrix with the class rooms in the rows and the timeslots in the columns. If a session is planned, the matrix will contain a session ID, otherwise it will contain 0.
- We also implemented different moves which define neighborhoods. One move simply shifts a session ID to a matrix element that equals 0. If a selected element differs from 0, it swaps the two sessions. Another move swaps all the scheduled sessions in a particular timeslot with any other timeslot.
- The last interface that a user of OpenTS needs to implement is the Objective Function interface. By implementing this interface the developer decides how the different hard and soft constraints will be evaluated. To ascertain the quality of a solution proposed by the tabu search algorithm (how many constraints are violated by the proposed solution), this solution needs to be evaluated by the objective function. At this moment only the hard constraints described in the previous section are evaluated.
- If an iteration is unsuccessful - no better solution is found in the considered iteration - the tabu length is increased. If, however, a better solution is found the length of the tabu list is decreased (until the lower limit of size 7 is reached). We opt to choose primes as tabu lengths. Switching between neighborhoods happens when the number of unimproving moves in one neighborhood exceeds a predefined value.

The described method allows to schedule a weekly timetable. As noted in Section 2 however, some sessions are only taught a few times during a semester.

4 Aggregation

To keep the problem size small, we opt not to construct twelve independent weekly timetables, which probably would not differ much from week to week. Instead we decided to carry out a pre-processing step. The idea is to group lectures which are not organized on a weekly basis into aggregate sessions, which we call ‘pillars’. If a lecture is only organized six times during a semester, the application will try to find a lecture that is also taught six times (or less). Fig. 1 shows a graphical representation of a semester consisting of twelve weeks. The ‘pillar’ in the middle of the figure is built up of two lectures organized in interleaved weeks (marked with and without a cross) that are each organized six times during a semester to the same class group. The evaluation method in this example only has to check the constraints for two consecutive weeks (assuming of course that these lectures are organized alternatingly).

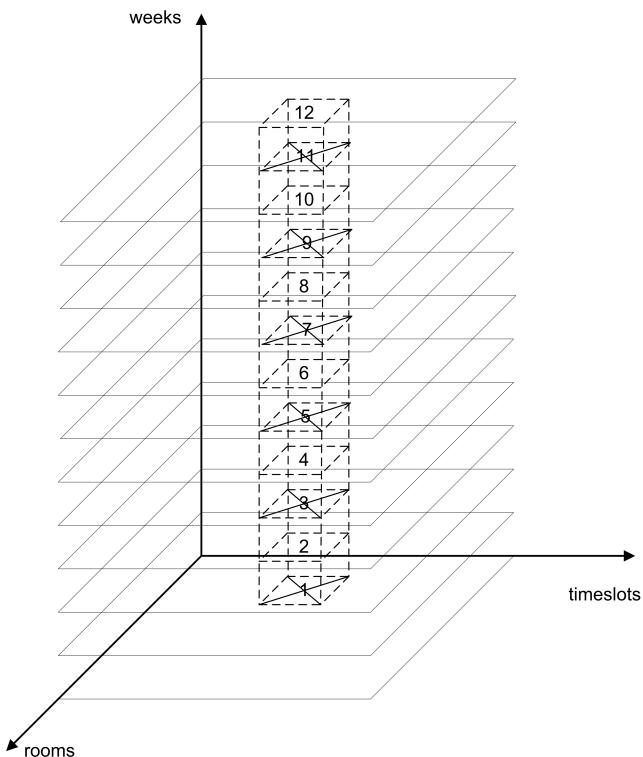


Fig. 1. Graphical representation of pillars. Sessions in the pillar are taught on a fortnightly basis.

The method is based upon the tiling approach that Kingston [13] successfully applies to Australian high school timetabling. He groups similar sessions and

treats them as one. We are also thinking about generalizing the pillars concept. In the case described above, the pillar can only be constructed for lectures organized to the same class group, using the same location and timeslot.

5 Results

The above described model is implemented and tested on real-world data. The preliminary results look rather promising.

6 Conclusion and Future Work

The presented model with the pillar representation offers a good quality method to solve large timetabling problems. Currently, it only deals with hard constraints. Remaining problems to solve are:

- investigating if the pillar representation can be generalized and applied to sessions that only have lecturers or students in common.
- taking into account soft constraints (e.g. avoid sessions on the last timeslot of the day, avoid free hours, take into account personal preferences of lecturers,...). One candidate approach is the linear numberings method [3,8] that was first introduced for employee timetabling. In that linear numberings method constraints are expressed in terms of eight numbering constraints and a corresponding template of numbers.

References

1. R. Alvarez-Valdes, E. Crespo, J.M. Tamarit: Design and implementation of a course scheduling system using Tabu Search, European Journal of Operational Research, Volume 137, Number 3, 512-523(12), 2002
2. V.A. Bardadym: Computer-Aided School and University Timetabling: The New Wave, Selected and Revised Papers of the 1st International Conference on Practice and Theory of Automated Timetabling, (PATAT 1995), Edinburgh, Springer LNCS 1153, 22-45, 1996
3. E.K. Burke., P. De Causmaecker, S. Petrovic, G. Vanden Berghe: Fitness Evaluation for Nurse Scheduling Problems, In Proceedings of Congress on Evolutionary Computation, CEC2001, Seoul, IEEE Press, 1139-1146, 2001
4. E.K. Burke, K.S. Jackson, J.H. Kingston and R.F. Weare: Automated Timetabling: The State of the Art, The Computer Journal, Vol. 40, No. 9, 565-571, 1997
5. E. Burke, G. Kendall, E. Soubeiga: A tabu-search hyperheuristic for timetabling and rostering, Journal of Heuristics, 9, 451-470, 2003
6. M.W. Carter and G. Laporte: Recent Development in Practical Course Timetabling, Selected and Revised Papers of the 2nd International Conference on Practice and Theory of Automated Timetabling, (PATAT 1997), Toronto, Springer LNCS 1408, 3-19, 1998

7. N. Custers, P. De Causmaecker, P. Demeester and G. Vanden Berghe: Semantic Components for Timetabling, Selected and Revised Papers of the 5th International Conference on Practice and Theory of Automated Timetabling, (PATAT 2004), Pittsburgh, Springer LNCS 3616, 17-33, 2005
8. P. De Causmaecker, P. Demeester, G. Vanden Berghe, B. Verbeke: Evaluation of Employee Rosters with the Extended Linear Numberings Method, Proceedings of UK PlanSIG, London, 125-133, 2005
9. F. Glover, M. Laguna: Tabu Search, Kluwer, Boston, 1997
10. R. Harder: OpenTS - Java Tabu Search
<http://www-124.ibm.com/developerworks/oss/coin/OpenTS/index.html>
11. A. Hertz: Tabu Search for Large Scale Timetabling Problems, European Journal of Operational Research, Vol. 54, 39-47, 1991
12. A. Hertz: Finding a feasible course schedule using Tabu search, Discrete Applied Mathematics, Vol. 35, 255-270, 1992
13. J. H. Kingston: A Tiling Algorithm for High School Timetabling, Selected and Revised Papers of the 5th International Conference on Practice and Theory of Automated Timetabling, (PATAT 2004), Pittsburgh, Springer LNCS 3616, 208-225, 2005
14. S. Petrovic and E. Burke: University Timetabling, Handbook of Scheduling: Algorithms, Models, and Performance Analysis (ed. J. Leung), CRC Press, Chapter 45, 45-1 - 45-23, 2004.
15. A. Schaefer: Local search techniques for large high-school timetabling problems, IEEE Transactions on Systems, Man, and Cybernetics- Part A: Systems and Humans, 29(4), 368-377, 1999

An Iterative Re-start Variable Neighbourhood Search for the Examination Timetabling Problem

Masri Ayob^{1,2}, Edmund K. Burke¹ and Graham Kendall¹

¹ ASAP Research Group, School of Computer Science & IT
University of Nottingham, Nottingham NG8 1BB, UK
{ekb, gxk}@cs.nott.ac.uk

² Faculty of Information Science and Technology
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
masri@finsm.ukm.my

1 Introduction

Producing good quality examination timetables is a difficult task which is faced by many academic institutions. Due to the complexity and the large size of the real-world university examination timetabling problems, it is difficult to obtain an optimal solution. Indeed, due to the complex nature of the problem, it is questionable if an end user would recognise a truly optimal solution.

Carter and Laporte [1] defined the exam timetabling as: “the assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes.”

In principle, the exam timetabling problem involves, assigning exams to timeslots subject to a set of hard and soft constraints. Hard constraints are rigidly enforced whilst soft constraints should be satisfied as far as possible. For example, exams which have common students have to be assigned to different timeslots (hard constraint). Wherever possible, examinations should be spread out over timeslots so that students have large gaps in between exams (soft constraint). Constraints vary among institutions and further discussion of exam timetabling constraints can be found in Burke et al. [2] and Carter and Laporte [1]. Timetables that satisfy all the hard constraints are called feasible solutions. Due to the complexity of the problem, it is not usually possible to have solutions that do not violate some of the soft constraints. Indeed, the evaluation of the cost function (how good the solutions are) is a function of violated soft constraints. A weighted penalty value is associated with each violation of the soft constraint and the objective is to minimise the total penalty value.

The exam timetabling problem can be modelled as a graph colouring problem (see Burke et al. [3, 4]). Usually, graph colouring heuristics, which order the events/exams based on an estimation of their difficulties, are used to construct the timetable. These include:

Largest degree first. This first schedules the exam that has the largest number of conflicts with other exams.

Colour degree. Exams with a greater number of conflicts with the exam that have already been scheduled have a higher priority of being scheduled next.

Saturation degree. Exams with fewer feasible slots are scheduled as early as possible.

Largest weighted degree. Exams with the higher number of students in conflict are scheduled earlier.

Largest enrolment. Exams with larger student enrolments are scheduled earlier.

Many approaches have been developed to solve the exam timetabling problem. These include graph heuristics (Burke et al. [5]), tabu search (Di Gaspero and Schaerf, [6]), evolutionary algorithms (Burke et al. [7]; Erben [8]; Côté et el. [9]), simulated annealing (Dowsland [10]), hyper-heuristics (Burke et al.[4]) etc. Extensive surveys and overviews on various approaches in solving timetabling problem can be found in Carter [11], Carter and Laporte [1], Schaerf [12], Burke and Petrovic [13] and Petrovic and Burke [14].

A survey by Carter [11] covers the exam timetabling research from 1964 until 1984, with Carter and Laporte [1] extending that survey. The later survey classified the methods used in solving the exam timetabling problem into four groups: cluster, sequential, meta-heuristic and constraint-based approaches. Cluster methods group the exams and then assign a timeslot to each group. Sequential approaches assign exams to timeslots consecutively. Constraint-based methods represent exams as a set of variables that have to be assigned to which values that represent resources such as rooms and timeslots, which satisfy some constraints (White [14]). Meta-heuristic approaches start with initial solution(s) and then apply search strategies to improve the solutions. Carter and Laporte [1] argued that most approaches only use simple constraints in solving the exam timetabling problems.

The exam timetabling problem is considered as an un-capacitated problem when room capacity is ignored. Whereas, a capacitated problem limits the number of students sitting exams in a slot but does not directly assign exams to specific rooms. The benchmark datasets (available at <ftp://ftp.mie.utoronto.ca/pub/carter/testprob>), which are used in this work, are an un-capacitated problem. These benchmark datasets were presented by Carter et al. [15].

This paper describes a computational approach to examination timetabling. A constructive heuristic based on saturation degree is used, followed by a local improvement heuristic based on a variant of variable neighbourhood search.

2 Variable Neighbourhood Search

Variable neighbourhood search (VNS) was introduced by Mladenović and Hansen [16]. VNS is a heuristic that is capable of exploring multi-neighborhood structures, hence it can explore distant neighbourhoods of the current solution (Mladenović and Hansen [16]). The shaking procedure in the basic VNS approach is a diversification factor whilst the local search will intensify the search to lead it converge to a local optimum. A local search (see Reeves and Beasley [17]; Aarts and Lenstra [18]) is applied repeatedly to obtain the local optima from the selected neighbouring solution.

There has recently been increasing interest in the *VNS* approach. For example, Avanthay et al. [19], developed an adaptation of *VNS* to solve the graph colouring problem with a Tabucol (a variant of tabu search) algorithm (Hertz and de Werra [20]) as a local search. They used three neighbourhood structures; these being vertex, class, and non-increasing neighbourhoods. Their *VNS* algorithm, however is not superior to the hybrid algorithm proposed by Galinier and Hao [21] that integrates a tabu search and a genetic algorithm. Some other works on *VNS* include Caporossi and Hansen [22], Morena Pérez et al. [23] and Fleszar and Hindi [24], which demonstrate that it is suitable across a number of different problem types.

3 Iterative Re-start Variable Neighbourhood Search

The basic *VNS* algorithm is a descent heuristic (Hansen and Mladenović, [25]), whilst our iterative re-start *VNS* (*IR-VNS*) is a descent-ascent heuristic. Let n_w , $w=1,2,\dots,W$, be a set of predefined neighbourhood structures, and $n_w(x)$ is the set of solutions in the w^{th} neighbourhood of x , $f(x)$ is the quality of solution x . W is the total number of neighbourhood structures to be used in the search. Our *VNS* algorithm is presented in fig. 1.

In our approach, we do not apply a shaking procedure before starting the local search since this might prolong the search time. Since exam timetabling problems have to deal with many constraints, finding good quality feasible solutions can be difficult and time consuming. The shaking mechanism within basic *VNS* (Mladenović and Hansen [16]) may cause the search to jump to a poor solution which may be difficult to escape from. Therefore, in our approach, we replace the shaking procedure by accepting the best neighbour (which might be worse) of the incumbent solution.

Since the initialisation strategy could influence the performance of the search algorithm (see Burke et al., [26]), especially when the search space is disconnected (which is a common case in exam timetabling problem), we use the saturation degree heuristic to iteratively construct incumbent solutions when the search becomes trapped in a local optimum.

At the improvement stage, we employ four neighbourhood structures as follows:

1. Steepest descent (N_1). A neighbour solution of x is generated by swapping all exams in the j^{th} slot with all exams in the $(j+k)^{th}$ slot. This local search returns the best neighbour after visiting all neighbours of x .
2. Free slot (N_2). A neighbour solution of x is generated by changing the slot of each exam to the best available slot. The local search returns the best neighbour (not necessarily an improved solution) after assigning the best new slot for each exam.
3. Swap two exams (N_3). A neighbour solution of x is generated by swapping one exam in the j^{th} slot with one exam in the $(j+k)^{th}$ slot. Again, this local search returns the best neighbour after visiting all neighbours of x .

Exponential Monte Carlo, EMCQ (N_4). This local search is adapted from Ayob and Kendall [27]. The EMCQ accepts an improved solution but probabilistically accepts a worse solution depending on the solution quality, search time and the time it is trapped in a local optimum. As in N_1 , a neighbour solution of x is generated by swapping all exams in the j^{th} slot with all exams in the $(j+k)^{th}$ slot. However, a trial solution will be accepted based on the EMCQ acceptance criterion. If it is accepted,

Step A: (Initialisation)

- (1) Select the set of neighbourhood structures n_w , $w=1,2,\dots,W$ that will be used in the search; choose a stopping condition and value of MAX;
- (2) Sorts the exams in decreasing number of exams in conflicts; Let N as the number of exams and E_i as the i^{th} exam where $i \in \{1,2,\dots,N\}$. Set $i=1$;

Step B: (Construction Stage)

- (1) Use E_i as a starting exam and construct an incumbent solution x using saturation degree heuristic with back-tracking and random slot assignment;
- (2) If this is the first iteration, then record the best obtained solution, $x_{best} \leftarrow x$ and $f(x_{best}) \leftarrow f(x)$;
- (3) Set Unimproved=0;

Step C: (Improvement Stage)

- (1) Set $w \leftarrow 1$;
- (2) Do
 - a). Exploration of neighbourhood. Find the best neighbour, x' from the w^{th} neighbourhood of x ($\mathcal{E}_w(x)$).
 - b). Accept the solution, $x \leftarrow x'$;
 - c). If $f(x') < f(x_{best})$ then $x_{best} \leftarrow x'$, $f(x_{best}) \leftarrow f(x')$ and Unimproved=0;
 - d). Else, Unimproved=Unimproved+1;
 - e). $w \leftarrow (w \bmod W) + 1$;

Until Unimproved =MAX or the stopping condition is met.
- (3) If Unimproved =MAX but the stopping condition is not met, then $i=(i \bmod N)+1$ and goto step B.
- (4) Otherwise, return the best obtained solution.

Note: MAX is the upper bound for Unimproved counter.

Fig. 1. The Proposed VNS algorithm for exam timetabling problem

we move to a new solution and the search continues by exploring a new neighbourhood (by continuing with the subsequent slot). Some moves might be performed before returning the best obtained solution to the VNS level. This is different from the N_1 neighbourhood, which only returns the best neighbour of one neighbourhood.

Currently, we use the following permutation: $\{N_1, N_2, N_1, N_3, N_1, N_4\}$. We repeatedly apply N_1 after exploring each neighbourhood structure because N_1 explores all neighbours of one neighbourhood. Since each local search explores different

neighbourhood structures, it might be worth visiting all neighbours in N_I after applying other local searchers. We are still investigating the effectiveness of repeatedly applying N_I , the order in which neighbourhood are explored and the upper bound value, MAX for an *Unimproved* counter. At this stage, we use $MAX=20$ and the above permutation.

4 Objective Function

We use a proximity cost as an objective function, which has been presented in Carter et al. [15], as follows:

$$\text{Minimise } F = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^N c_{ij} \cdot \text{proximity}(t_i, t_j)}{M} \quad (1)$$

Subject to:

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^N c_{ij} \cdot x(t_i, t_j) = 0 \quad (2)$$

$$x(t_i, t_j) = \begin{cases} 1 & \text{if } t_i = t_j; \\ 0 & \text{otherwise;} \end{cases} \quad (3)$$

Where,

N : is a number of exams;

M : is a number of students;

T : is a given number of available timeslot;

$C = (c_{ij})_{NxN}$: is a conflict matrix where each element denoted by c_{ij} , where $i, j \in \{1, 2, \dots, N\}$, is the number of students taking exam i and j ;

t_i : is a timeslot for exam i where $t_i \in \{0, 1, \dots, T-1\}$

$$\text{proximity}(t_i, t_j) = \begin{cases} \frac{2^5}{2^{|t_i - t_j|}} & \text{if } |t_i - t_j| \leq 5; \\ 0 & \text{otherwise;} \end{cases} \quad (4)$$

Equation 4 presents a weighted value (suggested by Carter et al. [15]) that reflect the cost of assigning exam i and j to timeslots. These being 0, 1, 2, 4, 8 and 16, where the cost is ‘0’ if the gap of slot for exam i and j is greater than 5. Equation (2) and (3) ensure a clash free timetable where each student will be sitting one exam at each timeslot.

5 Experiments and Results

In this work, we use the benchmark exam timetabling datasets presented in Carter et al. [15]. These datasets have been used by many researchers. However, due to some changes made by Carter et al. [15], there are two sets of benchmark dataset. Table 1 shows the latest version (updated on June 7, 2005) of Carter et al. [15].

Table 1. Characteristics of benchmark exam timetabling problems.

	Exams	Students	Slots
car-f-92	543	18,419	32
car-s-91	682	16,925	35
ear-f-83	190	1,125	24
hec-s-92	81	2,823	18
kfu-s-93	461	5349	20
lse-f-91	381	2,726	18
pur-s-93	2419	30,032	43
rye-f-92	486	11,483	23
sta-f-83	139	611	13
tre-s-92	261	4,360	23
uta-s-92	622	21,266	35
ute-s-92	184	2,750	10
yor-f-83	181	941	21

As discussed in section 3, this is ongoing research. Our preliminary experiment shows the following results (see table 2), which are comparable to the state-of-the-art approaches reported in the literature (Abdullah et al. [28]; Asmuni et al., 2005; Burke et al., [5, 4]; Burke and Newall [29]; Caramia et al.[30]; Carter et al. [15]; Di Gaspero and Schaerf, [6]).

Based on our preliminary result in table 2, we can see that our *IR-VNS* is capable of producing good quality solutions across all datasets. This shows that exploring various neighbourhood structures with iterative re-start is an effective search, which can avoid local optima and can jump to distant neighbourhood that might be more promising region.

Table 2. Results from our IR-VNS and the state-of-art approaches on benchmark exam timetabling problems based on the proximity cost..

	IR-VNS	Abdullah et al. [28]	Asmuni et al. [31]	Burke et al. [5]	Burke et al. [4]	Burke and Newall [29]	Caramia et al. [30]	Carter et al. [15]	Di Gaspero and Schaerf [6]
car-f-92	4.51	4.36	4.56	4.2	4.84	4.0	6.0	6.2	5.2
car-s-91	4.90	5.21	5.29	4.8	5.41	4.6	6.6	7.1	6.2
ear-f-83	36.28	34.87	37.02	35.4	38.19	37.05	29.3	36.4	45.7
hec-s-92	11.06	10.28	11.78	10.8	12.72	11.54	9.2	10.8	12.4
kfu-s-93	14.74	13.46	15.81	13.7	15.76	13.9	13.8	14.0	18.0
lse-f-91	12.08	10.24	12.09	10.4	13.15	10.82	9.6	10.5	15.5
pur-s-93	4.66	-	-	4.8	-	-	-	3.9	-
rye-f-92	10.67	8.74	10.35	8.9	-	-	-	7.3	-
sta-f-83	157.32	159.20	160.42	159.1	141.08	168.73	158.2	161.5	160.8
tre-s-92	8.92	8.13	8.67	8.3	8.85	8.35	9.4	9.6	10.1
uta-s-92	3.58	3.63	3.57	3.4	3.54	3.2	3.5	3.5	4.2
ute-s-92	26.36	24.21	27.78	25.7	32.01	25.83	24.4	25.8	29.0
yor-f-83	38.97	36.11	40.66	36.7	40.13	36.8	36.2	41.7	41.0

6 Conclusions

We have presented an iterative re-start variable neighbourhood search that has two stages; construction and improvement. In the construction stage, we employ a saturation degree graph colouring heuristic with back-tracking to construct an incumbent solution. We apply a variant of variable neighbourhood search to improve the incumbent solution. In our approach, we do not apply a shaking procedure before starting the local search. However, we diversify the search by accepting the best neighbour returned by the local search (which is not necessarily an improved solution). When the search becomes trapped in a local optimum, we jump to a distant solution space by reconstructing the incumbent solution using the saturation degree heuristic. Our preliminary results shows that our strategy is very promising, which can produce good quality solutions that are comparable to other published result.

References

1. Carter, M.W. and Laporte, G., (1996). Recent developments in practical examination timetabling. In: Burke and Ross (1996), 3-21.
2. Burke, E.K., Elliman, D.G., Ford, P. and Weare, R. F. (1996). Examination timetabling in British Universities – A survey. In: Burke and Ross, 76-92.
3. Burke, E.K., Elliman, D.G. and Weare, R.F., (1994). A university timetabling system based on graph colouring and constraint manipulation. Journal of Research on Computing in Education, 27(1), 1-18.

4. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R. (2006). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, in press.
5. Burke, E.K., Kingston, J. and de Werra, D. (2004a). Applications to timetabling. In: Gross, J. and Yellen, J. (eds.), *Handbook of graph theory*. Chapman Hall/CRC Press, 445-474.
6. Di Gaspero, L. and Schaerf, A., (2001). Tabu search techniques for examination timetabling. In: Burke, E.K. and Erben, W. (eds.), *Selected papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, 2079, 104-117.
7. Burke, E.K., Newall, J. and Weare, R. (1998). Initialization strategies and diversity in evolutionary timetabling. *Evolutionary Computation*, 6(1), 81-103.
8. Erben, W., (2001). A grouping genetic algorithm for graph coloring and exam timetabling. In: Burke, E.K. and Erben, W. (eds.), *Selected papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, 2079, 132-158.
9. Côté, P., Wong, T. and Sabourin, R., (2005) A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In: Burke, E.K. and Trick, M. (eds.), *Selected papers from the 5th International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, 3616, Springer-Verlag, 294-312.
10. Dowsland, K., (1998). Off the peg or made to measure. In: Burke, E.K. and Carter, M. (eds.), *Selected papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science, 1408, 37-52.
11. Carter, M.W., (1986). A survey of practical applications of examination timetabling algorithms. *Operations Research Society of America*, Volume 34 No 2, 193-202.
12. Schaerf, A., (1999) A survey of automated timetabling. *Artificial Intelligence Review*, 13, 87-127.
13. Burke, E.K. and Petrovic, S. (2002). Recent research directions in automated timetabling. *European Journal of Operational Research*, 140, 266-280.
14. Petrovic, S. and Burke, E.K., (2004), ch. 45- University timetabling. In: *The Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Leung, J. (edd.), CRC Press.
14. White, G. M., (2000). Constrained satisfaction, not so constrained satisfaction and the timetabling problem. *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Konstanz, Germany, 32-47.
15. Carter, M. W., Laporte, G. and Lee, S. Y. (1996) Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society* Volume 47 Issue 3, 373-383.
16. Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11), 1097-1100.
17. Reeves, C.R. and Beasley, J.E. (1995). Introduction, ch. 1. In: Reeves, C. R.(eds) *Modern heuristic techniques for combinatorial problems*, McGraw-Hill, 1-19.
18. Aarts, E. and Lenstra, J.K. (eds). (2003). *Local search in combinatorial optimization*. Princeton University Press.
19. Avanthay, C., Hertz, A. and Zufferey, N. (2003). A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151, 379-388.
20. Hertz, A. and de Werra, D. (1987). Using tabu search techniques for graph coloring, *Computing*, 39, 345-351.
21. Galinier, P. and Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3, 379-397.

22. Caporossi, G. and Hansen, P. (2004). Variable neighborhood search for extremal graphs 5. Three ways to automate finding conjectures. *Discrete Mathematics*, 276, 81-94.
23. Morena Pérez, J.A., Marcos Moreno-Vega, J. and Rodríguez Martín, I. (2003). Variable neighborhood tabu search and its application to the median cycle problem. *European Journal of Operational Research*, 151, 365-378.
24. Fleszar, K. and Hindi, K.H., (2004). Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155(2), 402-413.
25. Hansen, P. and Mladenović, N. (2001). Variable neighborhood search. *European Journal of Operational Research*, 130, 449-467.
26. Burke, E.K., Bykov, Y., Newall, J. and Petrovic, S., (2004b). A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, 36, 509-528.
27. Ayob, M. and Kendall, G., (2003). A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine, Proc. of the International Conference on Intelligent Technologies, InTech'03, Chiang Mai, Thailand, 132-141.
28. Abdullah, S., Ahmadi, S., Burke, E.K. and Dror, M. (2004). Applying Ahuja-Orlin's Large Neighbourhood for Constructing Examination Timetabling Solution , Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT), 413-419.
29. Burke, E. K. and Newall, J. P. (2003). Enhancing timetable solutions with local search methods. In: Practice and Theory of Automated Timetabling IV, E. K. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science Vol. 2740, Springer-Verlag, pp. 195-206.
30. Caramia, M., Dell' Olmo, P. and Italiano, G. F. (2001) New algorithms for examination timetabling. In: Algorithm Engineering 4th International Workshop, S. Näher and D. Wagner (Eds), Lecture Notes in Computer Science, Volume 1982, Springer-Verlag, 230-241.
31. Asmuni, H. and Burke, E.K. and Garibaldi, J.M., Fuzzy Multiple Heuristic Ordering for Course Timetabling, in Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI05), pp. 302-309, London, UK, 5-7 September 2005

A Simulated Annealing Hyper-heuristic for University Course Timetabling

Ruixin Bai¹, Edmund K. Burke¹, Graham Kendall¹
Barry McCollum²

¹ Automated Scheduling, Optimisation and Planning (ASAP) Research Group
School of Computer Science & IT, University of Nottingham
Nottingham, NG8 1BB, UK
`{rzb, ekb, gxk}@cs.nott.ac.uk`

² Department of Computer Science, Queen's University Belfast
Belfast, BT7 1NN, UK
`b.mccollum@qub.ac.uk`

1 Introduction

The university course timetabling problem involves assigning a given number of events (including lectures, seminars, labs, tutorials, etc) into a limited number of timeslots and rooms subject to given set of constraints. Two primary hard constraints are that no student should be assigned two events in one timeslot and that capacity and features of rooms should satisfy the requirement of the event. Other constraints can be different from one university to another. For example, some universities might want the timetable to be constructed so that there is a good separation between the courses that a student attends, while other universities may prefer to have consecutive courses.

Timetabling is a well-known difficult combinatorial problem. Several techniques have been used to automatically generate university timetabling problems, including graph colouring heuristics (Burke et al., 2004), tabu search (Costa, 1994; Schaefer, 1996), simulated annealing (Thompson and Dowsland, 1996; Kostuch, 2004), evolutionary algorithms (Burke et al., 1998) and case-based reasoning (Burke et al., 2006a). Hyper-heuristic approaches have recently been applied to timetabling problems (Burke et al., 2003; Burke et al., 2006b). In (Burke et al. 2003), a tabu search based hyper-heuristic was applied to both a nurse rostering problem and a university course timetabling problem to demonstrate the increased level of generality of the method. In their approach, the hyper-heuristic dynamically ranks a set of heuristics according to their performance in the search history. A tabu list was incorporated to prevent the selection of some heuristics at certain points in the search. At each iteration, the hyper-heuristic keeps applying the highest "non-tabu" heuristic to the current solution until the stopping criterion is met. Competitive results have been obtained on both problems when compared with other state-of-the-art techniques. In (Burke et al., 2006b), a case-based reasoning hyper-heuristic system was proposed for the course timetabling problem. The system differs from other case-based reasoning systems in that case-based reasoning was used to predict the best heuristic methods that can be

used to produce a good quality solution rather than finding a solution for the problem directly. The experimental results showed that the system can perform intelligently and effectively in the production of automated timetables.

In this research, we propose a simulated annealing hyper-heuristic approach (see fig. 1) for the university course timetabling problem. The simulated annealing hyper-heuristic manages a set of neighbourhood functions or heuristics and dynamically bias the selection of these heuristics. This approach has been successfully applied to a shelf space allocation problem (Bai and Kendall, 2005). It is hoped that the algorithm will either produce better results than the current proposed approaches or will reduce the computational time while generating good quality solutions.

2 Problem Description

In this research, we will study a course timetabling problem that was introduced in (Socha *et al.*, 2002). The problem can be described as follows:

Given a set of events e_i ($i = 0, \dots, n$) and a number of rooms r_j ($j = 0, \dots, m$) with each room having f types of features. Each event is attended by a given number of students and the total number of students is K . The aim of the problem is to assign every event e_i to a timeslot t_k ($k = 1, \dots, 45$) and a room r_j so that the following *hard constraints* are satisfied:

1. No student should be assigned to more than one event at a timeslot;
2. The room assigned to an event should have sufficient capacity and all the features required by the given event;
3. No more than two events can be scheduled in one room at a timeslot.

The objective of the problem is to minimise the number of students involved in the following soft constraint violations:

1. A event is scheduled at the last timeslot of the day;
2. A student has only one event in a day;
3. A student has more than two consecutive events.

In reality, the course timetabling problem is often much more complex (McCollum, 1998). Additional constraints will be considered in subsequent research to those presented here.

3 Simulated Annealing Hyper-heuristics

3.1 Hyper-heuristics

The aim of hyper-heuristics is to develop a reusable, generic optimisation approach for a range of different problems and different problem instances. Hyper-heuristics can be defined as “heuristics to choose heuristics”. Hyper-heuristics search the solution space indirectly by operating on heuristics. A hyper-heuristic makes use of a set of diverse domain-dependent heuristics or neighbourhood functions and strategically

changes their preferences during the local search in order to adapt to different situations and problem instances (Burke et al., 2003; Ross, 2005).

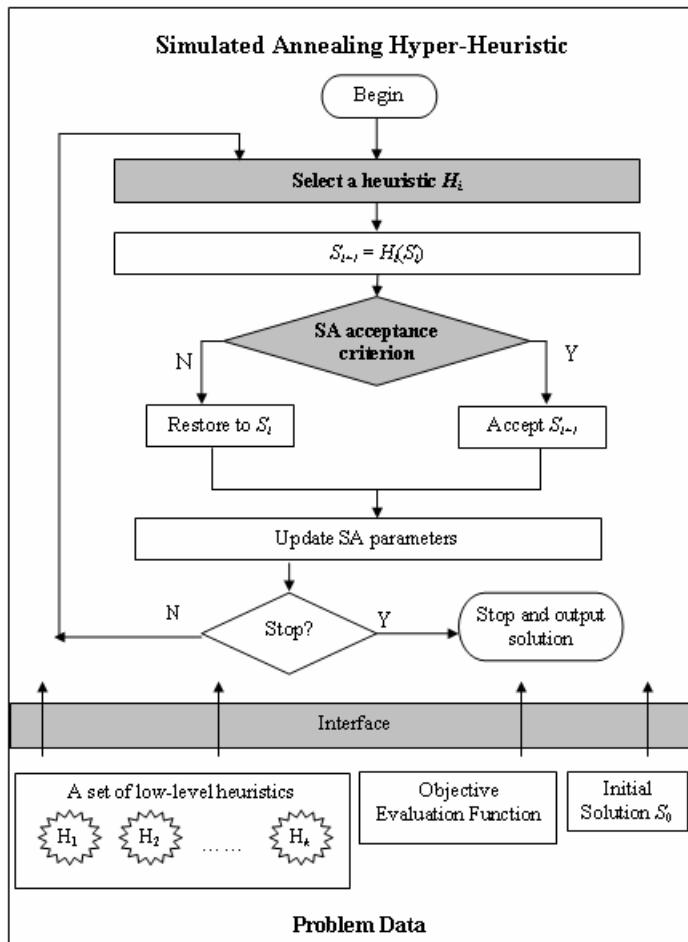


Fig. 1. The framework of simulated annealing hyper-heuristics for a maximisation problem

3.2 Simulated Annealing Hyper-heuristics

The proposed simulated annealing hyper-heuristic algorithm is a modified version of the algorithm in (Bai and Kendall, 2005), and is based on the following assumptions and observations.

1. A heuristic is selected probabilistically rather than deterministically. The reason for this is based on the empirical conclusion that probabilistically biasing the candidate solutions in SA is more beneficial than using a deterministic method (Tovey, 1988). A deterministic heuristic selection approach may be not suitable

for the simulated annealing hyper-heuristics due to the probabilistic characteristic of simulated annealing.

```

Set initial temperature  $t_s$ , stopping temperature  $t_f$  and total iterations  $K$ ;
Generate an initial solution  $s_0$ ;  $t=t_s$ ;
Define a set of heuristic  $H_i$  ( $i=0, \dots, n$ ), assign appropriate weight  $w_i$  to
each heuristic  $H_i$ ;
Do
  Select a heuristic ( $H_i$ ) based on probability  $p_i = w_i / \sum_{i=1}^n w_i$  ;
  Generate a candidate solution using heuristic  $H_i$ ;
  Let  $\delta_i$  stand for the difference in the evaluation function between s
  and  $s'$ ;
  If  $\delta_i > 0$ 
     $s=s'$ ;  $w_i = w_i + k$  ;
  else if  $\delta_i = 0$  and a new solution is created
     $s=s'$ ;  $w_i = w_i + \epsilon$ 
  else if  $\delta_i = 0$  and no new solution is created
     $w_i = w_i - \epsilon$ 
  else if  $\delta_i < 0$  and  $\exp(\delta_i/t) < \text{random}(0,1)$ 
     $w_i = w_i - k$  ;
  if  $w_i > w_{\max}$ 
     $w_i = w_{\max}$ 
  if  $w_i < w_{\min}$  ,
     $w_i = w_{\min}$ 
Loop until stopping criteria are met

```

Fig. 2. Pseudo-code of the simulated annealing hyper-heuristics

2. To bias the selection of heuristics, each heuristic is associated with a weight that reflects their importance at the current stage. During the search, these weights are dynamically changed based on the performance of their corresponding heuristics.
3. The mechanism to change the weights of heuristics is a *penalty-reward* strategy. That is, the weight of a heuristic is increased if it produces a better solution and decreased otherwise. However, for those heuristics that cannot improve the evaluation function, we distinguish between the heuristics that generate new solutions and those that do not. We observe that during the search, although some heuristics cannot improve the solution directly, they are still useful in creating some intermediate situations, from which the optimal solution (or a good quality solution) could be reached. Hence in this system, we give a minor positive score to those heuristics which could transfer the state of the solution but could not

improve the objective value. Meanwhile, we penalise those heuristics which could neither improve the current solution nor generate a new solution.

The pseudo-code of the algorithm can be described in figure 2.

4 Application

To apply the proposed simulated annealing hyper-heuristics, we need to design a set of problem-dependent heuristics. We shall use the heuristics that were used in (Burke et al., 2003; Abdullah et al., 2005). The algorithm will then be tested on the benchmark problems that was introduced by (Socha et al., 2002).

References

- 1 Abdullah, S., Burke, E. K. and McCollum, B., Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling. In the Proceedings of MIC 05: The 6th Meta-heuristic International Conference, Vienna, Austria, 22-26 Aug, 2005.
- 2 Bai, R. and Kendall, G., "An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics," in Ibaraki, T., Nonobe, K., and Yagiura, M. (eds.) Metaheuristics: Progress as Real Problem Solvers - (Operations Research/Computer Science Interfaces Series, Vol. 32) Berlin, Heidelberg, New York: Springer, pp. 87-108, 2005.
- 3 Burke, E. K., Causemacker, P. D., and Vanden Berghe, G., "Applications to Timetabling," in Gross, J. and Yellen, J. (eds.) Handbook of Graph Theory Chapman Hall/CRC Press, pp. 445-474, 2004.
- 4 Burke, E. K., MacCarthy, B. L., Petrovic, S. and Qu, R., Multiple-retrieval Case-based Reasoning for Course Timetabling Problems. Journal of Operations Research Society, 57(2): 148-162, 2006a.
- 5 Burke, E. K., Newall, J. P. and Weare, R. F., Initialisation Strategies and Diversity in Evolutionary Timetabling. Evolutionary Computation Journal (special issue on Scheduling), 6.1: 81-103, 1998.
- 6 Burke, E. K., Kendall, G. and Soubeiga, E., A Tabu-Search Hyperheuristic for Timetabling and Rostering. Journal of Heuristics, 9: 451-470, 2003.
- 7 Burke, E. K., Petrovic, S. and Qu, R., Case Based Heuristic Selection for Timetabling Problems. Journal of Scheduling, 9(2): 99-113, 2006b.
- 8 Costa, D., A Tabu Search Algorithm for Computing an Operational Timetable. European Journal of Operational Research, 76: 98-110, 1994.
- 9 Kostuch, Philipp, "The University Course Timetabling Problem with a 3-Phase Approach," in Burke, E. K. and Trick, M. (eds.) The Practice and Theory of Automated Timetabling V - Lecture Notes in Computer Science, Vol. 3616 Springer-Verlag, pp. 109-125, 2004.
- 10 McCollum, B., "The Implementation of a Centrally Computerised Timetabling System in a Large British Civic University," in Burke, E. K. and Carter, M. W. (eds.) Selected Papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT97), Lecture Notes in Computer Science Vol. 1408 Springer-Verlag, pp. 237-253, 1998.

- 11 Ross, Peter, "Hyper-Heuristics," in Burke, E. K. and Kendall, G. (eds.) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* Springer, pp. 529-556, 2005.
- 12 Schaerf, A., "Tabu Search Techniques for Large High-school Timetabling Problems," in Franz, A. and Kitano, H. (eds.) *Proceedings of the Thirteenth National Conference on Artificial Intelligence* AAAI Press, pp. 363-368, 1996.
- 13 Socha, K., Knowles, J. and Samples, M., A Max-min Ant System for the University Course Timetabling Problem. *Proceedings of the 3rd International Workshop on Ant Algorithm, ANTS 2002, Lecture Notes in Computer Science* 2463: 1-13, 2002.
- 14 Thompson, G. L. and Dowsland, K. A., Variants of Simulated Annealing for the Examination Timetabling Problem. *Annals of Operations Research*, 63: 105-128, 1996.

A Tiling Approach for Fast Implementation of the Traveling Tournament Problem

Amotz Bar-Noy and Douglas Moody

City University of New York Graduate Center

New York, NY

amotz@sci.brooklyn.cuny.edu, dmoody@citytech.cuny.edu

1 Introduction

Sports in society today are played by millions of individuals at the professional, scholastic and amateur levels. The success of the leagues and tournaments often lies in the ability to generate a “good” schedule. Each league or tournament has a variety of constraints and objective measures to be used in the determination of a good schedule. Availability of venues, the order of opponents, travel time and distance are but of a few of the myriad of issues considered by the sports league scheduler.

The Traveling Tournament Problem (TTP), documented by Easton et al. in [4], describes a typical sports scheduling challenge. Specific instances and records can be found in [14]. The TTP is a double round robin tournament to be played by n teams over $(2n-2)$ periods or weeks, where each team plays every period (we do not consider the “mirrored” version of the problem). Three unique constraints of the TTP are:

1. Maximum “Road Trip” of three games: each team can play at most 3 consecutive games away from the team’s home site before playing again at the home site. A *road trip* is defined as one or more consecutive games played away from the team’s home site, before returning home again. It is assumed that a team starts and ends the season from its home site.
2. Maximum “Home Stand” of three games: each can play at most 3 consecutive games at its home site. A *homestand* is defined as one or more consecutive games played at the team’s home site.
3. Repeater Rule: a team can not play an opponent away in time period k and then home in time period $k+1$, or vice versa.

The TTP seeks to minimize the distance traveled by each team. A distance matrix is used to calculate the distance from home to each opponent, and the distance between consecutive opponents. This calculation is done for each road trip, with the total being the *schedule distance*. The selection of the opponents and their order on the road trip is critical, while homestands have no bearing on the distance calculation. Effective development and placing of road trips will yield good solutions to the TTP.

The following sections describe related work and our general 3-phase approach, followed by a detailed description of each phase’s techniques and steps. The final two sections present results to date, and future work.

2 Related Work

The initial approaches to the TTP problem, and its more general round-robin tournament problem, centered on constraint and integer programmer approaches. Variations of models using this approach can be found in [5], [6], [7], [9], and [15]. The approaches create models with variables for each opponent pairing, home or away venue and usually other variables that represent the constraint being studied. As the number of teams grows slowly, these variables grow exponentially, significantly reducing their effectiveness with teams > 12 .

Heuristic approaches have also been used extensively. Regin in [11] looked at minimizing breaks (home game followed by an away game and vice versa) and patterns of home and away games. Pattern analysis and generation are also key components to the heuristic approach by Ribeiro in [12]. The pattern generation was then followed with local neighborhood swap techniques to improve the solution. Shen and Zhang in [13] generated patterns of team match-ups using a greedy approach.

As Henz described in [7], the TTP problem can also be viewed as a neighborhood search problem. Simulating annealing approaches in [1] and [10] take advantage of this perspective. Cardemil in [2] used tabu search logic.

Several of the above approaches relied on making several runs to find good starting neighborhoods. This step was followed by an extensive and costly local search effort searching for an optimal solution. We seek a fast implementation method that can come within 5-7% of the good solutions without using intensive resources through tiling. A tiling approach that constructed tiles and then timetabled was also used for course scheduling by Kingston in [8]. Courses are combined into tiles for forms (high school student years) for similar purposes, as our roadtrip tiles, and then timetabled into the scheduling grid.

3 Approach

Our approach is to model the road trips as “tiles”. Each tile will contain “blocks”, which represent individual games. A road trip of 3 opponents is considered as one tile, with three blocks. A teams’ schedule can be thought of as a series of tiles, with home games as spacers between the tiles. Figure 1 shows the scheduling grid and tiles for Team 1 and Team 2.



Fig. 1. Tile Placement for Teams 1 and 2 (shading cells indicate an away game)

For each team a set of tiles is created that seeks to minimize the distance traveled for a particular team without concern of any constraints involving other teams. These tiles are placed in a scheduling grid of n rows representing teams and $(2n-2)$ columns representing weeks.

As tiles are placed, other cells of the grid are filled in to keep the schedule consistent with the tile placement. When there are no tiles remaining that can be placed, the tiles are broken into their component blocks, and placed as allowable by TTP constraints. If not all blocks can be placed, the block placement is backtracked in an attempt to find a solution. The backtracking may reach back to the tile placement step, causing reassessments of tiles. If all blocks can be placed, a solution is generated, its distance calculated. Backtracking is again employed to find additional solutions.

4 Phase I – Tile Creation

The creation of the tiles is done on a team by team basis. For each team a minimum spanning tree (MST) is created by upon the Prim algorithm described in [3]. The algorithm begins with the selection of a root node, or in our approach a team. All distance edges in the distance matrix, defined in the problem, are searched for the smallest edge, which has a vertex of a team in the tree, and a vertex of a team not in the tree. This edge is then added between the two teams. For the first branch, we are finding the nearest opponent of the root team. The second edge is the nearest team to the root team, or its first opponent. Edges that are added to opponent will suggest the two vertices or teams will be on the same road trip or tile. Two edges both with having the root node as a vertex, suggest that the two opponents of the root team will be on different road trips. Figure 2 presents an example MST for the team Pittsburgh (PIT) in the nl6 problem in [4].

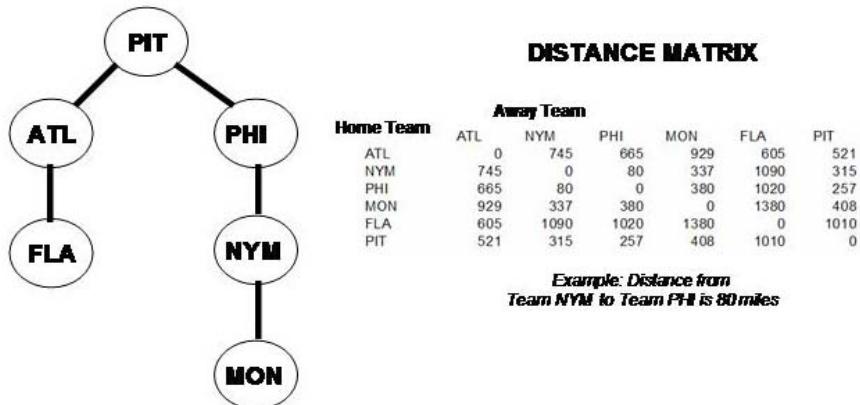
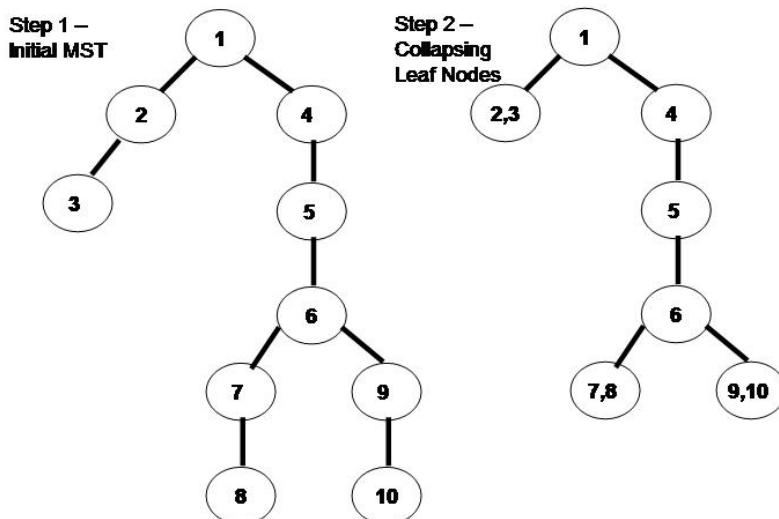


Fig. 2. MST for PIT in nl6 and the accompanying distance matrix.

Figure 2 suggests that the team PIT should have 2 road trips or tiles – one with ATL and FLA, and the other with PHI, followed by NYM and MON. If these two road trips are traveled by the team, the team will have the optimal minimum distance. This does not imply the league overall will have optimal minimal distance, but rather just this team.

We use a tree collapsing algorithm to create tiles from the tree structure. Separate trees are created with each tree having the root node of a team. The collapsing approach merges child nodes into their parent node. When the parent has 3 teams, a tile is made. When the parent must decide among its children, a greedy method is used, to pick the best set of 3 teams from the parent and children. Figure 3 provides a sample collapsing process.



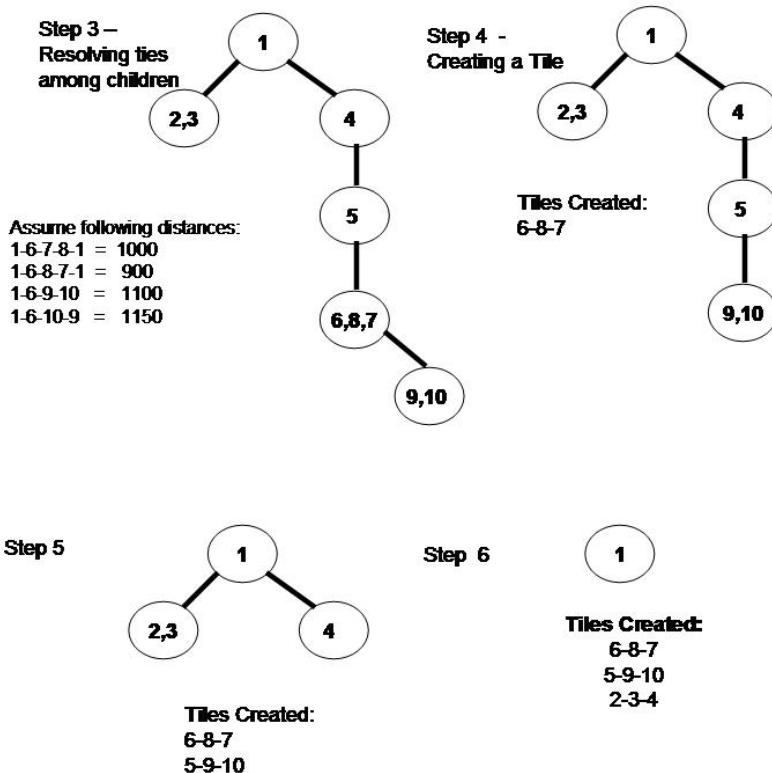


Fig. 3. Creation of Tiles through collapsing of the tree

The tile creation process creates ceiling($(n-1)/3 \times n$) tiles. Note that all tiles have 3 blocks, with the possible exception of the last tiles created for the team. At the root node, if both children have a weight of two nodes, two tiles of 2 blocks are created for each child.

5 Phase II – Tile Placement

The tile placement process places tiles, in team order by highest distance schedule, on the scheduling grid. All the tiles for a team are considered together. If necessary a tile is “rotated”, or its opponents are re-ordered. Switching opponent 1 and 3 within the tile does not affect the distance of the tile, however other switches do impact the distance. Only switches resulting in a 10% or less impact are acceptable.

The tile is moved through the schedule week by week. After all weeks have been tried, the blocks within the tile (games) are rotated in order to find an acceptable placement. After each placement, consistency checks are made to ensure the league schedule meet all the TTP constraints.

When no more tiles can be placed, Phase III is executed, which breaks the unplaced tiles in individual blocks. Backtracking is then performed to rearrange the blocks and look for additional solutions. During the backtracking, some tiles are placed in a “tabu” like status, so they are ignored temporarily giving less costlier tiles the chance to be placed earlier.

One noteworthy aspect of our approach is that tiles are never broken and then formed into new tiles, referred to as *reconfiguration*. This process involves breaking 2 tiles in their component blocks, and regrouping the blocks into two new tiles.

6 Phase III – Block Placement

Phase III is similar to other solutions focused on constraint programming solutions ([5],[7],[9]). The phase breaks all remaining unplaced tiles into individual blocks. These blocks, or games, are now placed in the scheduling grid one by one. Backtracking is used when a set of placed blocks can no longer lead to a solution. Multiple solutions can be found for the set of placed tiles through this phase. Other works have used a constraint programming package, such as ILOG, to accomplish this task.

7 Results

The first set of results examines the success of MST collapsing algorithm to generate valid tiles, based upon solutions in [4]. For comparison purposes, we looked at the solution records for leagues with teams of 6,8 and 16. When multiple away games appear in the solution schedule, a tile can be created (deduced) to represent those games. We compared the blocks of these tiles with those that are generated form the MST collapsing algorithm. The results are shown in Figure 5.

These results show a high correlation between the roadtrips embedded in the solutions with the roadtrip tiles generated by the MST. For a low number of teams the percentage match is about 80% of the tiles consisting of over 85% of the games. In the higher number of teams, the number of tiles and associated games involved is about 60%. In this case, the tiles that are different only add 5-7% miles to the total distance.

The second set of results considers the generation of the solution set itself. For the small instances where $n = 6$, the tiling approach produced a distance of 24,102. When the number of allowable free blocks (the maximum number of blocks or games needed to be scheduled individually) for entering Phase III (versus immediate backtrack) was increased beyond $3n$, our approach achieved the current goal of 23,916. This parameter ensures that sufficient tiles have been used before attempting individual block scheduling. This parameter could be relaxed, since the number of teams was small enough not to materially affect the length of phase III processing. Processing times for both solutions was less than 10 minutes on a 1.63 GHz computer with 1 Gig of memory, executing a Visual Basic custom application.

Solution Name	Total Solution Tiles	Total Games Solution Tiles	MST Complete Matching Tiles	MST 2 Partial Matching Tiles	% of MST Matching Tiles incl. Partial	% of MST Matching Games in Tiles
Easton – 6 teams	11	28	6	3	82%	86%
Easton – 8 teams	18	48	13	2	72%	90%
Cardil – 16 teams	83	211	25	27	63%	61%
Zhang – 16 teams (8/6/02)	83	219	17	40	69%	60%

Fig. 4. Comparison of tiles deduced from existing solutions versus MST collapsed tree generation

For $n = 8$ teams, a distance of 41,957 was achieved in under 30 minutes. This compares favorably to the existing best solution of 39,721, a 5.6% difference. During the tiling approach execution, 4,360 solutions were found. For 10 teams, the tiling approach achieved a distance of 62,916 compared to the existing best solution of 59,436, a 5.8% difference. The result took nearly an hour, however a solution of 68,204 was found within minutes. For 16 teams, Phase III processing needs further efficiencies to complete the single block processing inherent in that step.

8 Conclusion and further work

The MST collapsing approach, coupled with tile processing looks promising for producing very good, but not optimal, solutions in quick fashion. Our work to date shows that with a custom application program, we can create tiles and generate solutions within a short time, that come within 5-7% of existing solution records. As our Phase III constraint processing logic comes closer to commercial packages, our processing times (currently concentrated in Phase III processing) will decrease.

Improvement to our process will focus on consistency checks at the team level. Stronger analysis of the home and away pattern of a team at various points, can highlight inconsistencies, enabling necessary tile placement backtracking to occur earlier in the process. Also, Phase III processing may be replaced by a call to a constraint programming language such as ILOG.

References

1. Anagnostopoulos, A., Michel, L., Van Hentenryck, P., Vergados, Y. (2003) *A simulated annealing approach to the traveling tournament problem*, Proceedings CPAIOR'03, Montreal.
2. A. Cardemil. *Optimizacion de fixtures deportivos: Estado del arte y un algoritmo tabu search para el traveling tournament problem*. Master's thesis

- sis, Universidad de Buenos Aires, Departamento de Computacion, Buenos Aires, 2002.
3. Cormen, Thomas H. et al. Introduction to Algorithms. pp. 570-573. Boston: McGraw-Hill, 2001.
 4. Easton, K., Nemhauser, G., Trick, M. (2001) *The traveling tournament problem: description and benchmarks*, in: Proceedings CP'01, Lecture Notes in Computer Science 2239, Springer, 580-585.
 5. Easton, K., Nemhauser, G., Trick, M. (2003) *Solving the traveling tournament problem: a combined integer programming and constraint programming approach*, in: E. Burke and P. De Causmaecker (eds.), PATAT 2002, Lecture Notes in Computer Science 2740, Springer, 100-109.
 6. Henz, M. (1999) *Constraint-based round robin tournament planning*, in: D. De Schreye (ed.), Proceedings of the International Conference on Logic Programming, Las Cruces, New Mexico, MIT Press, 545-557.
 7. Henz, M. (2004) *Playing with constraint programming and large neighborhood search for traveling tournaments*, Proceedings PATAT 2004, Pittsburgh, USA.
 8. Kingston, J. (2004) *A tiling algorithm for high school timetabling*, Proceedings PATAT 2004, Pittsburgh, USA.
 9. Leong, G. (2003). Constraint programming for the traveling tournament problem. www.comp.nus.edu.sg/henz/students/gan_tiwaw_leong.pdf
 10. Lim, A., Zhang, X. (2003) *Integer programming and simulated annealing for scheduling sports competition on multiple venues*, Proceedings MIC 2003.
 11. Regin, J.-C. (2001) *Minimization of the number of breaks in sports scheduling problems using constraint programming*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 57, 115-130.
 12. Ribeiro, C.C., Urrutia, S. (2004) *Heuristics for the mirrored traveling tournament problem*, Proceedings PATAT 2004, Pittsburgh, USA.
 13. Shen, H., Zhang, H. (2004) *Greedy Big Steps as a Meta-Heuristic for Combinatorial Search*. The University of Iowa AR Reading Group, Spring 2004 Readings. goedel.cs.uiowa.edu/classes/AR-group/04spring.html
 14. Trick, M.A. Challenge Traveling tournament instances. Online document at <http://mat.gsia.cmu.edu/TOURN/>. Last update as of writing: January 18, 2006
 15. Trick, M.A. (2003) Integer and constraint programming approaches for round-robin tournament scheduling, in: E. Burke and P. De Causmaecker (eds.), PATAT 2002, Lecture Notes in Computer Science 2740, Springer, 63-77.

Understanding the Role of UFOs Within Space Exploitation

Camille Beyrouthy¹, Edmund K. Burke¹, J. Dario Landa-Silva¹, Barry McCollum^{2,3}, Paul McMullan^{2,3}, and Andrew J. Parkes^{1*}

¹ School of Computer Science & IT, University of Nottingham,
Nottingham NG8 1BB, UK.

{ cbb, ekb, jds, ajp }@cs.nott.ac.uk

² Queen's University of Belfast, Belfast, BT7 1NN, UK

{ b.mccollum, p.p.mcmullan }@qub.ac.uk

³ Realtime Solutions Ltd, 21 Stranmillis Road, Belfast, BT9 5AF
b.mccollum@realtimesolutions-uk.com

In this paper, we are concerned with understanding the efficient planning and management of teaching space, such as lecture rooms, within universities. There is a perception that such space is a rather scarce resource. However, some studies have revealed that in many institutions it is actually chronically under-used [3, 4]. Specifically, overall space-usage efficiency is measured by the “utilisation” (U), which is basically the percentage of available “seat-hours” that are exploited:

$$\text{Utilisation}, U = \frac{\text{used seat-hours}}{\text{total seat-hours available}} \quad (1)$$

It is also standard practice to measure the overall frequency, F ,

$$\text{Frequency}, F = \frac{\text{used time-slots}}{\text{total time-slots}} \quad (2)$$

and the room occupancy, O ,

$$\text{Occupancy}, O = \frac{\text{used seat-hours within occupied rooms}}{\text{total seat-hours available within occupied rooms}} \quad (3)$$

The three measures U , F and O are not independent. If all the rooms were the same size, then, directly from the definitions, we would have $U = FO$, and a similar relationship continues to hold when we have rooms of different sizes.

Surprisingly, in practice, rooms are often occupied only half the time ($F \approx 50\%$), and even when in use they are often only half full ($O \approx 50\%$), with the result that utilisations of 20-30% are not uncommon. The ‘Higher Education Funding Council for England’ (HEFCE) has reported low utilisations, and two of the authors have commercial experience of such low utilisations from their work with Realtime Solutions Ltd [3, 4].

Naturally, many institutions would like to improve this situation in order to reduce costs, improve services, or to permit teaching space to be converted to other uses. Also, for long-term capacity planning it is necessary to incorporate

* Contact Author (ajp). Authors listed alphabetically.

excess capacity in order to compensate for the expected low utilisations. Naturally, this is expensive, and we want to be able to ensure that spare capacity is well-engineered. However, such better management is hampered because there does not appear to be a good understanding of why low utilisations happen in the first place. This motivates our two main goals:

1. to understand the factors leading to low utilisations.
2. to develop methods to choose excess capacity that is more cost-efficient: aiming to reduce the teaching space that needs to be provided, whilst not increasing the risk of it turning out to be inadequate

To model the domain, we start from a simple event allocation problem. The goal is to select events so as to maximise the utilisation yet permit an assignment of events to rooms that satisfies the following standard hard constraints:

1. the size of an event must not exceed the room capacity
2. the number of events allocated to a room must not exceed the number of time-slots, as events cannot share room time-slots.

In this model the utilisation can be optimised in polynomial time [2]. However, on using real data for rooms and courses (obtained from one building of a university in Sydney, Australia) it was clear that this model gave unrealistically high values of utilisation (around 80-95%). This suggested that a model based purely on space issues is inadequate for real-world universities.

Moreover, in reality, event allocation usually takes place within the context of many constraints on locations and timings of events. Accordingly, we also included within our model objectives that are intended to provide a simplified approximation/abstraction of real timetabling issues; in particular the use of a conflict matrix between events, and a location penalty for placing events in rooms that belong to different departments. Note that the inclusion of the conflict matrix means that the polynomial time methods can no longer be used, and instead we use local search together with simulated annealing.

On exploration of the resulting multi-objective trade-off surfaces, we find that the utilisation can be forced down to much more realistic levels, in the range of 20-40%. The results support the hypothesis that the location and timetable penalties have the potential to dramatically drive down utilisations, and are a reasonable candidate to explain low utilisations in the real world.

Now let us return again to the issue of planning future capacity. We take the point of view that a (tentative) set of courses effectively form a “request for a given number of seat hours”, and hence correspond to a request for a given level of utilisation. In the absence of low utilisations, then we would be confident that as long as we request no more than 100% of the available seat-hours then we would be able to satisfy all of the request. However, this is no longer true when utilisation is expected to be lower than 100%. Hence, we set up the methodology to answer the following question

“Under what conditions is a request for utilisation fully satisfiable?”

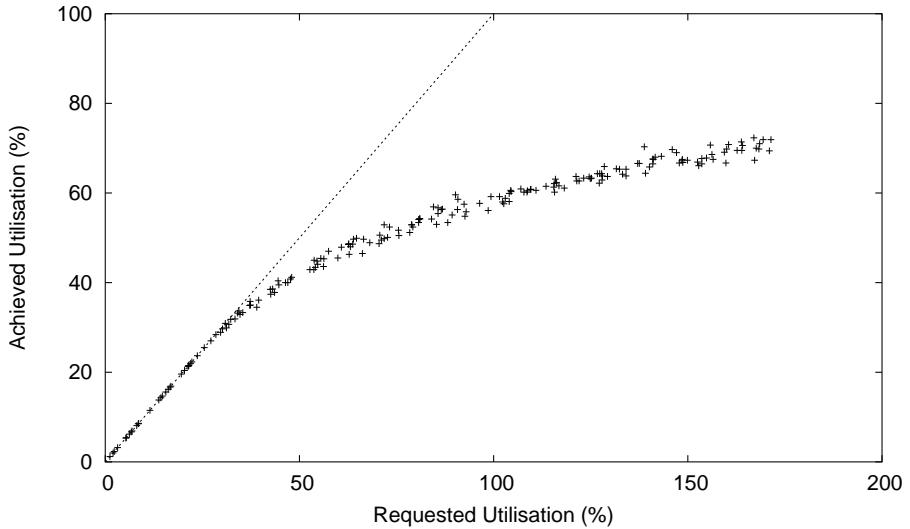


Fig. 1. An example of Requested Utilisation (U_R) vs. Achieved Utilisation (U_A). The line $U_R = U_A$ is given for reference purposes.

We studied this question by taking a wide variety of randomly-selected subsets of our real set of events, and for each subset finding the maximum achievable utilisation. A representative example is shown in figure 1 for the achieved utilisation plotted against the requested utilisation. From this and other experiments, we obtain the following results.

Firstly, the values of achieved utilisations for given corresponding requests, tend to be “grouped around the mean”: The variation between points near to some requested value is small. This implies that properties of the system are statistically predictable.

Secondly, we see a threshold phenomenon on the utilisation U . There is a “critical value”, U_C , for the requested utilisation, U_R , that demarcates a sharp division between regions in which the answer is “almost always yes” and those of “almost always no”. (In the case of Figure 1 we have $U_C \approx 30\%$.) We then have two distinct regions:

SAFE: $U_R < U_C$. Requests for the seat-hours are almost always totally satisfied.

UNSAFE: $U_R > U_C$. Requests for the seat-hours are almost never totally satisfied. Even in the cases when there are sufficient seat-hours available, it turns out that the oversupply is very unlikely to be usable.

These results are typical of those in threshold phenomena; perhaps best-known within the context of random graphs [1]. For example, the chromatic number of random graphs is similarly predictable. The threshold behaviour has

an important implication. When planning course offerings we cannot assume that we can simply count seat-hours, but must realise that we are unlikely to be able to rely upon using more than some predictable critical utilisation, and this will (almost) inevitably mean that some of the events will need to be dropped.

Our work suggests that progress in space management and planning will arise from an integrated approach. Firstly, combining purely space issues with restrictions representing an aggregated or abstracted version of key constraints such as timetabling or location. And secondly, also performing statistical studies to reveal underlying threshold phenomena.

References

1. B. Bollobas. *Random Graphs*. Academic Press, London, England, 1985.
2. Michael W. Carter and Craig A. Tovey. When is the classroom assignment problem hard? *Operations Research*, 40(1):28–39, 1992.
3. B. McCollum and P. McMullan. The cornerstone of effective management and planning of space. Technical report, Realtime Solutions Ltd, Jan 2004.
4. B. McCollum and T. Roche. Scenarios for allocation of space. Technical report, Realtime Solutions Ltd, 2004.

New concepts in neighborhood search for permutation optimization problems

Wojciech Bożejko¹ and Mieczysław Wodecki²

¹ Institute of Engineering Cybernetics, Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
email: wbo@ict.pwr.wroc.pl

² Institute of Computer Science, University of Wrocław
Przesmyckiego 20, 51-151 Wrocław, Poland
email: mwd@ii.uni.wroc.pl

1 Introduction

For many strongly NP-hard combinatorial optimization problems a natural representation of a solution is a permutation. Practical approaches to solve such problems are local search algorithms based on the neighborhood's search. Well chosen neighborhood is one of the key elements, which affects efficiency of this method. Classic neighborhoods have polynomial number of elements and they are generated by single moves. Neighborhoods with an exponential number of elements have been successfully applied for a few years. We introduce the neighborhood (and its properties – neighborhood graph, diameter, etc.), which has application in the best local search algorithms. This neighborhood belongs to very large scale neighborhood class (VLSN) and it is generated by swap multimoves. We present a theorem which states that any permutation (solution) can be transformed into any other permutation by execution at most two swap multimoves (that is the diameter of the neighborhood graph equals 2). The second theorem of this paper states that the problem of determining an optimal swap multimove in the neighborhood is NP-hard. Applying this neighborhood (and the algorithms of its exploration) to local search algorithms allows us to obtain the best known results for the most difficult scheduling problems considered in literature (single machine total weighted tardiness problem, flow shop and job shop problems).

In this paper we propose a neighborhood generated by a composition of all the swap moves, where supports of the permutation connected with these moves are disjoint - that is different swaps of a multiswap does not change the same elements of the permutation (they are commutative). Such a neighborhood is very large (has an exponential number of elements). Determining its optimal element is an NP-hard problem.

One of the characteristics of the neighborhood structure is the diameter of the corresponding graph. We will also prove, that the diameter of the multiswap neighborhood graph is 2. Applying this neighborhood to local search algorithms allows us to obtain the best known results for the most difficult scheduling

problems considered in literature (e.g. single machine total weighted tardiness problem [2], flow shop [6] and job shop problems [7]).

2 Swap multimoves

We consider a combinatorial optimization problem with a permutational representation of a solution. Let $\mathcal{I} = \{1, 2, \dots, n\}$ be a set of n elements (enumerated by numbers from 1 to n). By $S(n)$ we mean a set of all permutations of the set \mathcal{I} .

Permutation $\sigma \in S(n)$ is called *involution* [8], if it is inverse to itself, i.e. $\sigma^2 = \epsilon$, where ϵ is the identity permutation. It is simple to see, that if $\sigma \in S(n)$ is an involution, then an inverse permutation σ^{-1} is an involution too.

Fact 1 *Every involution is a composition of pair-independent (with disjoint supports) transpositions.*

Conclusion 1. If $\pi \in S(n)$ and m is a swap move, then executing of this move generates a permutation $\beta = m(\pi)$. If the move m swaps an element $\pi(i)$ with $\pi(j)$, therefore it is easy to see, that $\beta = m(\pi) = \pi\alpha$, where $\alpha = \begin{pmatrix} 1 & 2 & \dots & i-1 & i & i+1 & \dots & j-1 & j & j+1 & \dots & n \\ 1 & 2 & \dots & i-1 & j & i+1 & \dots & j-1 & i & j+1 & \dots & n \end{pmatrix}$ is a transposition. So the move m can be identified with a transposition α . Therefore Fact 1 follows, that an involution is a composition of swap moves (and we call it a multiswap).

Theorem 1 [3] *For any permutations $\pi, \delta \in S(n)$ there exists involutions $\alpha, \beta \in S(n)$ such, that $\pi\alpha\beta = \delta$.*

Conclusion 2. From Theorem 1 and Fact 1 it follows that for any permutation $\delta \in S(n)$ there exist involutions $\alpha = \alpha_1\alpha_2\dots\alpha_l$ and $\beta = \beta_1\beta_2\dots\beta_l$, where $\alpha_s\beta_s$ ($s = 1, 2, \dots, l$) are independent cycles and such that $\delta = \alpha\beta$. The method of constructing of both involutions is precisely described in the proof of the theorem.

Lemma 1 *The diameter of the neighborhood graph corresponding to multimove swap neighborhood is 2.*

Proof. We consider any two permutations $\pi, \delta \in S(n)$ - nodes of the neighborhood graph $G = (V, A)$. From Theorem 1 it follows, that $\pi\alpha\beta = \delta$, where permutations α, β are involutions. Because $\pi\alpha \in \mathcal{N}(\pi)$, then $(\pi, \pi\alpha) \in A$ (that is the length between them is 1). Similarly, $\delta \in \mathcal{N}(\pi\alpha)$, therefore $(\pi\alpha, \delta) \in A$. It follows that the diameter of the multiswap neighborhood graph is 2. ■

Remark 1. In one of the advanced heuristic methods – path relinking – for two permutations π and δ a permutation γ , lying on a path between π and δ is constructed. Because $\pi\alpha\beta = \delta$, where α, β are involutions, so permutation $\gamma = \pi\alpha$, which is generated from π by an involution (multimove) α , lies on the path between π and δ . It is possible to construct such an involution α that permutation γ is in the required range of distance to π . In particular construction

of involutions α, β (see Conclusion 2) can be successfully applied to diversify the calculations, as a fully deterministic method. From construction involution with many transpositions it follows that permutation $\pi\alpha$ is at a long distance (counted as a number of swap moves) from π .

Remark 2. For any permutation $\pi \in S(n)$ the set of all permutations $S(n)$ can be divided into three subsets (orbits): a) $\{\pi\}$, b) $\{\pi\alpha : \alpha \in S(n)$ and α is an involution }, c) $\{\pi\alpha\beta : \alpha, \beta \in S(n)$ and α, β are involutions } $\}.$ The dynasearch neighborhood from the paper [4] is generated by single multimoves and these neighborhoods are subsets of the set defined in point b).

Remark 3. The neighborhood based on swap multimoves is of very large-scale (the number of its elements is asymptotically $\frac{1}{\sqrt{2}} \left(\frac{n}{e}\right)^{\frac{n}{2}} e^{\sqrt{n}-\frac{1}{4}}$, see [8]). We will prove that the problem of finding the optimal multiswap in the multimove swap neighborhood is equivalent to finding an optimal traveling salesman path in a certain graph.

Theorem 2 [3] *The problem of determining an optimal multiswap in the multimove swap neighborhood is NP-hard.*

Remark 4. In the works [6],[7] and [2] multimoves are applied to intensify and diversify the calculations. Because these moves are compositions of independent moves, therefore one can estimate the goal function of the permutation generated by these moves with good precision.

3 Application: The permutation flow shop scheduling

Garey, Johnson and Seti [5] show that for three machine flow shop problem ($F|3|C_{\max}$) is strongly NP-hard. The best available branch and bound algorithms are those of Lageweg, Lenstra and Rinnooy Kan [9]. Their performance is not entirely satisfactory however, as they experience difficulty in solving instances with 20 jobs and 5 machines. Various local search methods are available for the permutation flow shop problem. A very fast tabu search algorithms is proposed by Grabowski, Wodecki [6]. Multimoves are applied to diversify calculations in this algorithm. We have checked how they influence computations time and values of solutions.

The implementation of the tabu search algorithm TSGW [6] was tested on benchmark instances taken from the OR-library [1] and compared with reference results from this library. For comparison, the results of the TSGW algorithm and the TSGWnoMM algorithm (i.e. TSGW without multimoves) are presented in Table 3. The results are shown of two groups of columns: one is for the TSGW algorithm, the other is for the TSGWnoMM algorithm. Time in seconds (CPU) and percentage relative deviation (PRD) to the reference solutions are presented in Table 1.

Table 1. Quality test results.

$n m$	TSGW		TSGWnoMM	
	CPU	PRD	CPU	PRD
20 5	0.0	0.00	0.0	0.02
20 10	0.3	0.03	0.3	0.08
20 20	0.6	0.07	0.6	0.11
50 5	0.4	-0.08	0.4	0.05
50 10	0.9	-0.29	0.8	0.17
50 20	2.3	0.13	2.2	0.26
100 5	0.6	-0.03	0.6	0.14
100 10	1.4	-0.21	1.3	0.32
100 20	5.0	-0.68	4.5	0.19
200 10	4.4	-0.19	4.1	0.06
200 20	8.5	-1.12	7.9	-0.04
500 20	11.2	-0.81	10.7	0.23
all	2.96	-0.26	2.62	0.12

On the basis of results presented in Table 1 we can say, that computations times are almost the same. However definitely different are average relative deviations to the reference solutions. Average relative percentage deviation (PRD) for the TSGW (with multimoves) is -0.26, however for the TSGWnoMM (without multimoves) average PRD is 0.12. Applying of the multimoves cause about 3 times decreasing of the average PRD.

References

1. Beasley J.E., OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society 41, 1990, 1069-1072.
2. Bożejko W., M.Wodecki, Parallel algorithm for some single machine scheduling problems, Automatics vol. **134** 2002 81-90.
3. Bożejko W., M.Wodecki, On the theoretical properties of swap multimoves, Operations Research Letters, Elsevier (in press).
4. Congram, R.K., C.N. Potts, S.L. Van de Velde, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS Journal on Computing vol. **14** No. 1 2002 52-67.
5. Garey M.R., D.S. Johnson, R. Seti, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research, 1, 1976, 117-129.
6. Grabowski J., M. Wodecki, A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion, Computers and Operations Research **31** 2004 1891-1909.
7. Grabowski J., M. Wodecki, A very fast tabu search algorithm for the job shop problem. In: Rego C., Alidaee B., editors. Adaptive memory and evolution; tabu search and scatter search, Dordrecht, Kluwer Academic Publishers 2005.
8. Knuth D.E., The art of computer programming, Vol. 3., second edition, Addison Wesley Longman, Inc. 1998.
9. Lageweg B.J., J.K., Lenstra, A.H.G. Rinnooy Kan, A General Bounding Scheme for the Permutation Flow-Schop Problem, Operations Research, 26, 1978, 53-67.

Scheduling Sport Leagues using Branch-and-Price

Dirk Briskorn

Institute for Production und Logistics, University of Kiel, Germany,
briskorn@bwl.uni-kiel.de

Sports league scheduling is a hard combinatorial optimization problem. There is a vast field of requests arising in real world problems, e.g., organizational, attractiveness and fairness constraints. A single round robin tournament (SRRT) can be described as a league of a set T of n teams (n even) to be scheduled such that each team plays exactly once against each other team and such that each team plays exactly once per matchday (MD) resulting in a set P of $n - 1$ MDs. Matches are carried out at one of opponents' stadiums. A team playing twice at home or twice away in two consecutive periods is said to have a break in the latter of both periods. The number of breaks is to be minimized. It is well known that at least $n - 2$ breaks must occur. We focus on schedules having the minimum number of breaks. Costs corresponding to each possible match are given and the objective is to minimize the sum of matches' cost. This can be formulated as a cost minimization IP as follows:

$$\min \sum_{p \in P} \sum_{i \in T} \sum_{j \in T: j \neq i} c_{i,j,p} x_{i,j,p} \quad (1)$$

s.t.

$$\sum_{p \in P} (x_{i,j,p} + x_{j,i,p}) = 1 \quad \forall i, j \in T : i < j \quad (2)$$

$$\sum_{j \in T: j \neq i} (x_{i,j,p} + x_{j,i,p}) = 1 \quad \forall i \in T, p \in P \quad (3)$$

$$\sum_{j \in T: j \neq i} (x_{i,j,(p-1)} + x_{i,j,p}) - br_{i,p} \leq 1 \quad \forall i \in T, p \in P^{\geq 2} \quad (4)$$

$$\sum_{j \in T: j \neq i} (x_{j,i,(p-1)} + x_{j,i,p}) - br_{i,p} \leq 1 \quad \forall i \in T, p \in P^{\geq 2} \quad (5)$$

$$\sum_{i \in T} \sum_{t \in P^{\geq 2}} br_{i,t} \leq n - 2 \quad (6)$$

$$x_{i,j,p} \in \{0, 1\} \forall i, j \in T : j \neq i, p \in P \quad (7)$$

$$br_{i,p} \in \{0, 1\} \forall i \in T, p \in P^{\geq 2} \quad (8)$$

$x_{i,j,p}$ is equal to 1 if and only if team $i \in T$ plays at home against team $j \in T$ at MD $p \in P$. Constraints (2) and (3) force the matches to form a SRRT. Equations (4) and (5) set $br_{i,p}$ to 1 if team i plays twice at home and away,

respectively, at MDs $p - 1$ and p . The overall number of breaks is restricted to be no more than $n - 2$ by constraint (6). The objective function (1) represents the goal to minimize the overall sum of cost of all matches. Costs can be interpreted in an abstract way here but it is not difficult to think of several applications having practical relevance. For example $c_{i,j,p}$ can be employed to represent the teams' neglected preferences to play home or away in p if match (i, j, p) is carried out. There are several papers covering models being equal or at least closely related to the one at hand, see [8], [3], [1], [7], and [4] for example. In [2] a basic SRRT problem covering (1), (2), (3), and (7) is proven to be NP-hard.

We exhibit an approach employing the well-known concept called column generation (CG) which enumerates the variables of a large-scale linear program implicitly. CG has been successfully implemented for graph coloring in [5]. Scheduling a SRRT is equivalent to edge-coloring a complete graph K_n with $n - 1$ colors. Our approach considers MDs as columns resulting in the pricing problem being a standard perfect-matching problem. Furthermore, we take home-away patterns (HAP) into account. A HAP can be represented by a string for each team i containing 0 in slot p iff i plays home at MD p . If we assign a HAP to each single team the pricing problem can be reduced to 2-dimensional assignment problems which can be solved efficiently by, e.g., the hungarian method.

This CG approach is employed within a branch-and-price framework. The branching concept underlies the idea to create sets of HAPs by branching on break-periods of a specific team. There are several properties of HAPs inducing a minimum number of breaks known from [6] which can be employed to reduce the number of branches to be evaluated.

We present the integer program representing the structural requirements of SRRTs and several additional constraints. Furthermore, we give possible extensions considering availability of stadiums, attractive matches, and fairness aspects. Details of the CG approach are outlined as well as the branching concept. Finally, we show that our algorithm outperforms CPLEX 9.0 by means of computational results and propose fields of future research.

References

1. T. Bartsch, Andreas Drexl, and Stefan Kröger, *Scheduling the professional soccer leagues of Austria and Germany*, Computers & Operations Research **33** (2006), 1907–1937.
2. Dirk Briskorn, Andreas Drexl, and F. C. R. Spieksma, *Round robin tournaments and three index assignment*, Working Paper (2006).
3. D. de Werra, *Geography, games and graphs*, Discrete Applied Mathematics **2** (1980), 327–337.
4. Andreas Drexl and Sigrid Knust, *Sports league scheduling: graph- and resource-based models*, Omega **to appear** (2006).
5. A. Mehrotra and M. A. Trick, *A column generation approach for graph coloring*, INFORMS Journal on Computing **8** (1996), 344–354.

6. R. Miyashiro, H. Iwasaki, and T. Matsui, *Characterizing feasible pattern sets with a minimum number of breaks*, Proceedings of the 4th international conference on the practice and theory of automated timetabling (E. Burke and P. de Causmaecker, eds.), Lecture Notes in Computer Science 2740, Springer, Berlin, Germany, 2003, pp. 78–99.
7. J. A. M. Schreuder, *Combinatorial aspects of construction of competition dutch professional football leagues*, Discrete Applied Mathematics **35** (1992), 301–312.
8. Michael Trick, *Integer and constraint programming approaches for round robin tournament scheduling*, Proceedings of the 4th international conference on the practice and theory of automated timetabling (E. Burke and P. de Causmaecker, eds.), Lecture Notes in Computer Science 2740, Springer, Berlin, Germany, 2003, pp. 63–77.

Solving Exam Timetabling Problems with the Flex-Deluge Algorithm

Edmund K. Burke, Yuri Bykov

Automated Scheduling, Optimisation and Planning Group,
School of Computer Science & IT, University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
`{ekb,yxb}@cs.nott.ac.uk`

In this abstract we present a new exam timetabling algorithm together with a set of results on the university exam timetabling problems from the University of Toronto collection, available at `ftp://ftp.mie.utoronto.ca/pub/carter/testprob/`. A number of recent papers have studied these problems e.g. Carter et al. [7], Caramia et al. [6], Casey & Thompson [8], Abdullah et al. [1], Burke et al. [2],[4]. We will compare the results of our new algorithm against these results.

In [4] and [5], we investigated a Great Deluge algorithm for exam timetabling. The basic algorithm was introduced by Dueck [11] and accepts a candidate solution if it satisfies the following conditions:

$$P' \leq B \quad \text{when } P < B \quad P' \leq P \quad \text{when } P \geq B \quad (1)$$

where P is the current penalty, P' is the penalty of the candidate solution and B is the current *upper limit* (called the “level”). At the beginning, B is equal to the initial penalty and with each step it is lowered by a decay rate (denoted by ΔB), which corresponds to the *search speed*. In [4], it was shown that the right choice of ΔB helps to fit the search procedure into an available time limit and that (unsurprisingly) longer searches generally produce better results.

In this paper, we propose an extension of the Great Deluge algorithm (which we call “Flex-Deluge”), where the acceptance of uphill moves depends on a “flexibility” coefficient k_f ($0 \leq k_f \leq 1$). The acceptance rules are outlined in Expression (2):

$$P' \leq P + k_f(B - P) \quad \text{when } P < B \quad P' \leq P \quad \text{when } P \geq B. \quad (2)$$

By varying k_f , it is possible to obtain an algorithm with characteristics of both the original Great Deluge ($k_f = 1$) and greedy Hill-Climbing ($k_f = 0$). This property is similar to that of the Peckish strategy (an intermediate between Hill-Climbing and Random Ordering) proposed by Corne and Ross in [9].

The proposed mechanism enables the search procedure to develop with an adaptive level of *strictness of acceptance* for each particular move. The method draws upon an idea of White & Xie [13], who suspended the *movement* of exams with low degree in order to leave more room for the movement of higher degree exams. The *degree* of an exam here is defined in terms of graph colouring (see [3]). Thus, when moving an

exam into a different timeslot we calculate the flexibility coefficient as a ratio of the exam's degree to the maximum degree.

A series of experiments has been carried out. The following three new features were added to the algorithm of Burke et al. [4]:

- Employing the flexible acceptance condition as described above.
- When a move causes an infeasible solution, the algorithm uses Kempe chains to repair infeasibility. The advantages of this technique for exam timetabling are highlighted by Thomson and Dowsland in [12].
- In addition, we also follow suggestions that are derived from the work of Di Gaspero [10]. The normal procedure is to re-allocate a randomly chosen exam to a new (also randomly chosen) timeslot. However, in this approach, in approximately 20% of the cases, we instead perform just the swapping of all the exams in two randomly chosen timeslots. The flexibility for this second type of move was chosen to be 0.5 (after a series of experiments).

The software was written in Delphi 7 and run on a PC Pentium 4 3.2 MHz. Each run lasted 5-10 hours while performing up to 2×10^9 moves. In [4] it was stated that this time is quite acceptable for exam timetabling (because in real world situations exam timetables are produced months before they are required) and there is no reason to reduce the time taken at the expense of the quality of solution.

We present the results of our algorithm on the eleven most commonly studied problems from the Toronto benchmark set. The identifiers and characteristics of the problems are presented in Table 1. Also, this table contains the comparison of our best results with a range of published ones, including the first results of Carter et al. [7], the original Great Deluge results [4] and the most successful results from other author's work.

Table 1. Published and our best results on benchmark problems

Dataset	Exams	Periods	Carter et al. (1996)	Caramia et al. (2001)	Burke & Newall (2003)	Casey & Thompson (2003)	Abdullah et al. (2004)	Burke et al. (2004)	Flex-Deluge
Car-s-91	682	35	7.1	6.6	4.6	5.4	5.21	4.8	4.42
Car-f-92	543	32	6.2	6.0	4.0	4.4	4.36	4.2	3.74
Ear-f-83	190	24	36.4	29.3	37.05	34.8	34.87	35.4	32.76
Hec-s-92	81	18	10.8	9.2	11.54	10.8	10.28	10.8	10.15
Kfu-s-93	461	20	14.0	13.8	13.9	14.1	13.46	13.7	12.96
Lse-f-91	381	18	10.5	9.6	10.82	14.7	10.24	10.4	9.83
Sta-f-83	139	13	161.5	150.2	168.73	134.7	150.28	159.1	157.03
Tre-s-92	261	23	9.6	9.4	8.35	8.7	8.13	8.3	7.75
Uta-s-92	622	35	3.5	3.5	3.2	-	3.63	3.4	3.06
Ute-s-92	184	10	25.8	24.3	25.83	25.4	24.21	25.7	24.82
Yor-f-83	181	21	41.7	36.2	36.8	37.5	36.11	36.7	34.84

The results produced by our method support the strength of the suggested approach. Also, they suggest that the effectiveness of the method is relatively higher for the large-scale problems. This also holds for the original Great Deluge exam timetabling method (see [4]).

References

1. Abdullah, S., Ahmadi, S., Burke, E. K., Dror, M.: Investigating Ahuja-Orlin's large neighbourhood search for examination timetabling. Accepted for publication in OR Spectrum, to appear 2006
2. Burke, E. K., Newall, J.: Enhancing timetable solutions with local search methods. In: Burke, E. K., De Causmaecker, P. (eds.): Selected papers from the 4th International conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science, Vol. 2740, Springer-Verlag, Berlin Heidelberg New York (2003) 344-354
3. Burke, E. K., Kingston, J., De Werra, D.: 5.6: Applications to Timetabling. In Gross, J., Yellen, J. (eds.): The Handbook of Graph Theory. Chapman Hall/CRC Press (2004) 445-474
4. Burke, E. K., Bykov, Y., Newall, J., Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Transactions, Vol. 36, No. 6 (2004) 509-528
5. Bykov Y.: Time-Predefined and Trajectory Based search: Single and Multiobjective approaches to exam timetabling. PhD thesis, the University of Nottingham, UK (2003)
6. Caramia, M., Dell'Olmo, P., Italiano, G.: New algorithms for examination timetabling. In: Naher, S., Wagner, D. (eds.): Algorithm Engineering 4th International Workshop, WAE 2000. Lecture Notes in Computer Science, Vol. 1982. Springer-Verlag, Berlin Heidelberg New York (2001) 230-241
7. Carter, M., Laporte, G., Lee, S.: Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society Vol. 47, (1996) 373-383
8. Casey, S., Thompson, J.: GRASPing the examination scheduling problem. In: Burke, E. K., De Causmaecker, P. (eds.): Selected papers from the 4th International conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science, Vol. 2740. Springer-Verlag, Berlin Heidelberg New York (2003) 232-246
9. Corne, D., Ross, P.: Peckish initialisation strategies for evolutionary timetabling. In: Burke, E. K., Ross, P. (eds.): Selected papers from the 1st International conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science, Vol. 1153, Springer-Verlag, Berlin Heidelberg New York (1996) 227-240
10. Di Gaspero, L.: Recolour, Shake and Kick: a recipe for the Examination Timetabling Problem" (abstract). In: Proceedings of the 4th International conference on Practice and theory of automated timetabling (2002) 404-407
11. Dueck G.: New optimisation heuristics. The great deluge algorithm and record-to-record travel. Journal of Computational Physics, Vol. 104 (1993) 86-92
12. Thompson, J.M., Dowsland, K.A.: Variants of simulated annealing for the examination timetabling problem. Annals of Operations research, Vol. 63 (1996) 105-128
13. White, G. M., Xie, B. S.: Examination timetables and tabu search with longer term memory. In: Burke, E. K., Erben, W. (eds.): Selected papers from the 3rd International conference on the Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science, Vol. 2079. Springer-Verlag, Berlin Heidelberg New York (2001) 85-103.

Examination Timetabling: A New Formulation

Edmund K. Burke^{1,3}, Barry McCollum^{2,3}, Paul McMullan^{2,3}, Rong Qu¹

¹ School of Computer Science and IT
University of Nottingham
Jubilee Campus, Nottingham NG8 1BB UK
{ekb, rxq}@cs.nott.ac.uk

² School of Computer Science, Queen's University
University Road, Belfast, Northern Ireland BT7 1NN
{b.mccollum, ppmcmullan}@qub.ac.uk

³ eventMAP Limited
21 Stranmillis Road, Belfast

Since the mid 1990's, with the implementation of increasingly flexible modular course structures in many UK 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 as to provide students with a maximum distribution of their exams throughout the examination session. Of course, we must also ensure that time and resources usage maximised. Universities, struggling with rising student numbers, more flexibility in choice and less time to examine, have increasingly relied upon automation of this task to produce efficient timetables which satisfy these constraints e.g. [1,2]. Although strong in some respects, unfortunately, many of the search methodologies currently described in the literature have some limitations in terms of potential application in a wide number of differing institutions.

The examination timetabling problem has long represented an area where new and exciting techniques have been trialed at an early stage of their development. This, in large part, is related to the inherently straightforward nature of the timetabling problem which can be expressed by the following. Students are examined over a designated time period, within a finite area of space, in such a way as to ensure they do not have two exams at the same time. This 'hard' constraint must be satisfied for a solution to be viable. The quality of the overall solution is measured by factors such as how well an individual's exams are distributed throughout the designated time period e.g. soft constraints. Both type of constraints, hard and soft, were documented in some detail for UK Universities in 1996 [1]. All subsequent research has been trialed on datasets which have been in existence from the middle part of that decade. The initiative described in this presentation sets out to update the situation with regard to the examination timetabling problem by investigating the changes in the problem along with providing new updated datasets for techniques to be subsequently trialed. It is well reported that there is a gap between theory and practice in scheduling research (eg [3]). A major contributor to the work presented in this abstract is our spin out company, EventMAP Limited. From a practitioner's point of view, the company has reported the steady increase in complexity of the examination problem over the last five years.

Overall, there are a number of goals that we intend to tackle in this work. Firstly, we aim to make available a number of new examination datasets complete with the all important space details. A major criticism of the work to date is that essential information relating to space usage has not been available for the purpose of allowing a true representation of the problem to be worked on. Although the capacitated examination timetabling problem has been investigated [4], this only served to place an upper limit on seats available at any particular time period. Crucial issues related to the number and sizes of rooms were absent. The anonymised datasets of real world scenarios will be made available to the research community via the web site at <http://www.cs.nott.ac.uk/~rxq/data.htm>, which will be dedicated to the datasets along with initial ‘diagnostic characteristics’ e.g. a conflict density matrix. In this way, we suggest that a master copy of the datasets can be held eliminating various discrepancies reported in the past [5]. This data provided by eventMAP Limited is necessary to understand the exact nature of the current real world issues within examination timetabling.

Secondly, we will introduce a new measure of solution quality which more accurately reflects the desired goals of the university sector. The issue of no room information being available with regard to the currently used datasets has meant that the optimisation function used to measure solutions has not incorporated all the necessary issues. Results using this newly introduced objective will be presented on the new university datasets. It is expected that this work will represent an important contribution in both updating and enabling the entire area of research within examination timetabling to move forwards.

Thirdly, information will be discussed with the purpose of determining the exact nature of this need and how the situation within UK Universities has changed and developed since the original investigation of Burke et al in 1996 [1]. Details from the institutions contributing to this research via the datasets will be gathered and presented with the aim of updating the type and amount of constraints that need to be taken into consideration when providing solutions.

Finally, eventMAP has shown that ‘best’ solutions to various datasets depend on the combination of different types of construction and improvement heuristics. This will be briefly discussed as a conclusion to the presentation.

References

1. Burke, E.K., D.G. Elliman, P.H. Ford, and R.F. Weare. (1996). “Examination Timetabling in British Universities – A Survey.” In E.K. Burke and P. Ross (eds.), *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st International Conference*, pp 76-90. Lecture Notes in Computer Science, Vol. 1153. Springer.
2. Burke, E.K., Eckersley, A., McCollum, B., Petrovic, S., Qu, R., Case Based Heuristic Selection investigation of Hill Climbing, Simulated Annealing and Tabu Search for Exam Timetabling Problems. Proceedings of the Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Gent, Belgium, 21-23 August 2002, ISBN 90-806096-1-7, pages 408-412.
3. EPSRC Document Review of Research Status of Operational Research in the UK, 2004.

4. Burke EK, Newall JP and Weare RF (1996a). A Memetic Algorithm for University Exam Timetabling. In Edmund Burke and Peter Ross, editors, *The Practice and Theory of Automated Timetabling*. Lecture Notes in Computer Science 1153. Springer, pp 3-21.
5. Qu R., Burke E.K., McCollum B., Merlot L.T.G., and Lee S.Y. The State of the Art of Examination Timetabling. Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham.

Progress Control in Variable Neighbourhood Search

Tim Curtois¹, Laurens Fijn van Draat², Jan-Kees van Ommeren³, and Gerhard Post²³

¹ ASAP, School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, UK

² ORTEC bv, Groningenweg 6K, 2803 PV Gouda, The Netherlands, www.ortec.com

³ Department of Applied Mathematics, University Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

1 Introduction

The methods of intensification and diversification are indispensable in successful meta heuristics for local search. Intensification corresponds in some sense to local optimisation; the neighbourhood of a solution is searched intensively for solutions which are better or have better opportunities. On the other hand, diversification tries to escape from (relatively small) neighbourhoods to solutions which might lead to better final results. A heuristic that is well aware of the intensification versus diversification problems, is the Variable Neighbourhood Search (VNS), see [2]. In this method, more than one neighbourhood structure is considered. After finishing intensification with respect to one neighbourhood, the heuristic diversifies to another neighbourhood. In this way one hopes to escape from poor local optima.

In this work we introduce a model to predict the quality of a neighbourhood. We use this model to identify ‘bad’ neighbourhoods and avoid searching them. We call this process ‘Progress Control’. Computational results are presented to show that progress control helps us finding better solutions in the same amount of time.

2 Problem setting

2.1 Alternating Neighbourhood Search

Our object of research is a specific type of VNS, in which there is a ‘small’ neighbourhood, and a ‘large’ neighbourhood. After reaching a local optimum with respect to the small neighbourhood, we move our attention to another solution in the large neighbourhood. From here we again perform local optimisation with respect to the new small neighbourhood. If we reach a better local optimum, we take this new solution as the starting point for further investigation, otherwise we revert to the old local optimum, and continue from there. We call this method ‘Alternating Neighbourhood Search’ (ANS).

In ANS it is normal that 99% of the calculation time is spent on local optimisation with respect to the small neighbourhood. A possibility to improve the performance of ANS is to detect, somehow, that it is useless to continue the current local optimisation. The most obvious way is to stop if, after some fixed amount of time, a certain quality has not been reached. In our experience, this method is not good enough: a bad solution that improves steadily might still be a promising one.

In this section we propose a model for the progress of the local optimisation process. With this model we follow the progress and try to predict which neighbourhoods are good and which are bad.

2.2 Modeling the progress

Suppose we follow the local optimisation and get an update of the result every τ milliseconds. With c_n we denote the cost after n periods. Furthermore we denote with $c = c_\infty$ the cost of the local optimum that eventually will be reached. Since we apply local optimisation, we know that $c_{n+1} \leq c_n$. The moves are selected randomly, hence it seems reasonable that the chance that a move at time t_n is successful is proportional to the difference $c_n - c$. Hence, for the expected value of the cost c_n we get

$$E(c_n - c) = \lambda(c_{n-1} - c).$$

Solving this recursion relation we obtain

$$E(c_n) = (c_0 - c)\lambda^n + c. \quad (1)$$

In continuous time we use $t_n = n \cdot \tau$ to get

$$c(t) = Ae^{Bt} + C, \quad (2)$$

which is more convenient to use. Here $A = c_0 - c$, $C = c$, and $e^{B\tau} = \lambda$.

Several assumptions underlie this model:

1. The number of good moves is proportional to the difference of the current score and the end score. In practice this is not correct: some violations of soft constraints can be removed by several moves and will probably found earlier. This implies that λ decreases with time (hence our early estimates are optimistic).
2. The cost change of a good move is always the same. In practice this is not correct: soft constraint violations with high costs are usually detected earlier, as a move can lift this high cost at introducing small costs for other soft constraints. Again early estimates are optimistic.
3. All moves take an equal calculation time. In practice this is not exactly the case, but if τ is relatively big, several hundreds of moves occur between t_n and t_{n+1} , and the number of these moves is with high probability close to average.

All these assumptions make that the predicted progress differs from the realised progress: the realised progress is much more irregular, especially in the beginning of the local optimisation. For this reason we use a rolling horizon for the progress controller, see section 2.3.

Although the model above is the most appealing model, we did not use it in our more extensive experiments. Preliminary experiments revealed that the approach in (2) is hard to scale: quite soon the exponential factor B is strongly negative, and C equals approximately the latest cost. To circumvent this problem we assume that the local optimisation will reach a perfect result of cost 0. Hence we try to approximate the progress with the curve

$$c(t) = Ae^{Bt}. \quad (3)$$

With this formula we can estimate t^* , the time needed before the local optimiser will reach the quality of the current best result. If this estimated time t^* is larger than a certain reference time T we stop the local optimisation. The assumption that the local optimisation will end with cost 0 is obviously not correct. But as before we take an optimistic point of view: if we would estimate the end cost to be higher, the time t^* would be larger, which implies that we would stop the process sooner.

2.3 Progress control

Using the (natural) logarithm in (3) leads to the linear function:

$$\log c(t) = \log A + Bt. \quad (4)$$

We use the least squares method to find the line $y = \alpha x + \beta$, approximating the points (x_k, y_k) where $x_k = t_k$ and $y_k = \log(c_k)$. A side effect of the least squares method is that big jumps in the costs have a relative large influence; since jumps are always downward, this will pull the curve (3) down. This seems good in our situation, where jumps correspond to an unstable situation, in which case some more patience seems appropriate.

We use our estimates of α and β to predict t^* each time we record a new value of c_n . For calculating t^* and taking the decision whether or not to stop we use three parameters p, f , and m :

- (p) We use a *rolling horizon*: we forget about early points, and estimate the future progress only based on the last p points.
- (f) We use a *start-up time*: for the first f points we record whether $t^* > T$ or not, but we never stop the process. If $f = \infty$ we perform ANS without progress control.
- (m) We use *forgiveness*: we do not necessarily stop directly if some $t_n^* > T$, but only if this happened more than m times in the last p checks. Clearly we will never stop before t_m ; hence it is useless to consider $f < m$. In particular if $m \geq p$, we do not stop at all.

We call a combination of these parameters (p, f, m) a *strategy*. Our main objective is to investigate the influence of a chosen strategy on the quality of the solution, assuming that we have a limited time at our disposal.

3 Results

The problem area we consider is personnel scheduling; the objects we try to schedule are shifts, which are fixed in time. The shifts have to be assigned to resources (employees), such that all hard constraints are satisfied, and the cost, resulting from trespassing soft constraints, is minimised. We use the Variable Neighbourhood Search as introduced in [1] as ANS. This algorithm is implemented in the advanced planning system HARMONY, developed by ORTEC for workforce management and scheduling.

We tested our progress control on five different datasets with different sizes. For each dataset we simulated 1000 runs and recorded the average values of some key figures for different strategies. In Table 1 we give per dataset the key figures for not applying any strategy (the lines with the ‘-’) and for the best strategy found. The size of the problems is indicated by the column ‘# Em.’.

Name	# Em.	p^*	f^*	m^*	Bad stop	Bad cont.	Good stop	Good cont.	Tries	Time	Dec.	(Dec./ Time) *1000
DataA	12	-	-	-	0	318	0	682	1.5	1090	243	223
		14	3	0	315	3	586	96	10.1	948	426	449
Data2	8	-	-	-	0	476	0	524	1.9	3105	14	4.6
		33	3	1	470	6	478	46	22	2122	28	13.2
Data1	46	-	-	-	0	639	0	361	2.7	245	10.9	44
		3	2	0	639	0	239	122	8.2	112	14.6	130
ORTEC1	16	-	-	-	0	704	0	296	3.4	515	503	975
		46	3	2	646	58	229	67	14.7	459	605	1319
ORTEC2	16	-	-	-	0	800	0	200	4.8	952	324	341
		33	9	4	748	52	116	84	11.9	760	472	622

Table 1. Best found strategies for five different datasets and the average values of the key figures.

The columns ‘Bad stop’ and ‘Bad cont.’ show the number of ‘bad’ neighbourhoods that are stopped or continued by our strategy. A ‘bad’ neighbourhood is a neighbourhood where the local optimisation eventually does not reach a better solution than the best known so far. We see that the optimal strategies filter out almost all of the bad neighbourhoods, just as we hoped. However, in the columns ‘Good stop’ and ‘Good cont.’ we see that our strategies also stop a lot of good neighbourhoods. As a result we see in the column ‘Tries’ that our strategies need 3 to 10 times more neighbourhoods to find a better solution than not applying any strategy. But still, in the column ‘Time’ we can see that it takes our strategies *less* time to find a better solution. For Data1 we even see that we gain more than 50% in time! The column ‘Dec.’ shows the difference between the score to beat and the score of the solution found after the first improvement. Here we see that the improvement in score is 20-100% higher for our strategies compared

to not applying progress control. This is caused by the fact that we are likely to skip neighbourhoods that give only a small improvement.

We can conclude that our progress control finds *better* solutions and that it finds them *faster*. The combination of these effects is shown in the last column. Of course it will be very hard, if not impossible, to find the optimal strategy *before* the optimisation process. But we found out that most strategies are an improvement compared to not applying progress control at all, so finding a *good* strategy is not so hard.

4 Conclusions

We defined progress control as the process of estimating the progress of a local optimisation in a neighbourhood and using these estimates to stop searching ‘bad’ neighbourhoods. The results show that progress control is a good idea in our Shift Scheduling problem. Since we didn’t use any knowledge of the problem or even of the VNS we used, we believe that it can be applicable to many versions of Iterated Local Search.

Finally, we like to stress that we do not pretend that our model predicts the expected value of c_∞ ; as can be noted in equation (3) we do not even try this. In fact, we are not even interested in the progress itself. We only try to predict whether or not a local optimisation will end up in a solution with a better score than the best known so far.

References

1. E. Burke, T. Curtois, G. Post, R. Qu, B. Veltman, *A hybrid heuristic and variable neighbourhood search for the nurse rostering problem*, Proceeding of the 5th international conference on the Practice and Theory of Automated Timetabling (PATAT 2004), pp. 445-446 (2004).
2. N. Mladenović, P. Hansen, *Variable Neighborhood Search*, Computers & Operations Research **24**, pp. 1097–1100, (1997).

Scheduling with Soft CLP(*FD*) Solver*

Tomáš Černý, Hana Rudová

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

Timetabling problems often consists from various requirements which can not be satisfied together [11]. Unsatifiable requirements can be handled within optimization criteria as *soft constraints*. Such soft constraints may not be satisfied if there are some contradictions. The remaining hard constraints must be still satisfied. The set of hard and soft constraints can be naturally expressed using constraint programming [3]. Unfortunately there is no system available which would allow to use both hard and soft constraints together. There are various systems implementing soft constraints [2] but none of them allows to combine efficient constraint propagation algorithms for classical constraint satisfaction problems together with the propagation for soft constraints.

Our implementation of the system for timetabling problem at Purdue University [9] was able to use both hard and soft constraints together. Hard constraints were available from SICStus Prolog CLP(*FD*) library [1] and soft constraints were implemented using the Soft CLP(*FD*) Solver [8]. This solver for soft constraints includes the specific set of soft constraints needed for the Purdue University timetabling problem. Our current intent is to generalize this proposal and implement the Soft CLP(*FD*) Solver as an open extendable library able to solve a wide class of problems.

The new solver allows to define soft constraints as it is shown in the course timetabling example in Figure 1. Hard constraint `disjoint2` ensures that all

```
(1) class_timetabling( TimesAndRooms ) :-
    ...
(2)     disjoint2( [class(Time, Duration, Room, 1) | Rest] ),
(3)     serialized( Times, Durations ),
(4)     Time in 7..8 @ discouraged,
(5)     TimeForLecture #< TimeForSeminar @ preferred,
(6)     soft_serialized(TimeForSeminar1,TimeForSeminar2,TimeForSeminar3),
    ...
(7)     labeling( TimesAndRooms ).
```

Fig. 1. Course timetabling example

* This work is supported by the Ministry of Education of the Czech Republic under the research intent No. 0021622419.

classes in the list do not overlap in time and space. Another hard constraint `serialized` allows for example to teach all classes of the same teacher at different times. The soft constraint (4) discourages placement of the class identified by its starting `Time` at seven or eight o'clock. Soft constraints can specify desired or undesired relations between classes (5). Soft global constraint `soft_serialized` allows to express that seminars of the same course should be preferably taught at different times.

While the original solver is based on partial forward checking [4], the new proposal allows to consider AC* and NC* consistency [6]. The variables in soft constraints are called *preference variables* and they are implemented using the attributed variable [5]. The attribute of each preference variable stores the current cost for each value present in the domain of the variable. Each preference variable is also a standard domain variable which allows to include it in any (hard) constraint of the CLP(*FD*) library. The *lower bound* of the solution is represented as a domain variable. Potential violations of soft constraints contribute to this cost and backtracking occurs when the lower bound of the current partial solution is greater than some existing upper bound.

The current implementation contains the basic unary and binary soft constraints. We plan to implement some soft global constraints based on principles described in [7]. There are some proposals for soft global cardinality constraints [10] we would like to study. These could be very interesting for solving of the employee timetabling problems. We also intend to use the solver for machine scheduling problems where various soft constraints are specified by users submitting tasks to the problem.

References

1. Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programming*. Springer-Verlag LNCS 1292, 1997.
2. Simon de Givry. Soft constraint satisfaction problem, 2005. <http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/>.
3. Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
4. Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
5. Christian Holzbaur. *Specification of Constraint Based Inference Mechanism through Extended Unification*. PhD thesis, University of Vienna, 1990.
6. Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1–2):1–26, 2004.
7. T. Petit, J.-C. Régin, and C. Bessiere. Meta-constraints on violations for over constrained problems. In *12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI00)*, pages 358–365.
8. Hana Rudová. Soft CLP(*FD*). In Susan Haller and Ingrid Russell, editors, *Proceedings of the 16th International Florida Artificial Intelligence Symposium, FLAIRS-03*, pages 202–206. AAAI Press, 2003.

9. Hana Rudová and Keith Murray. University course timetabling with soft constraints. In Edmund Burke and Patrick De Causmaecker, editors, *Practice and Theory of Automated Timetabling, Selected Revised Papers*, pages 310–328. Springer-Verlag LNCS 2740, 2003.
10. Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming (softening global constraints). In *6th International Workshop on Preferences and Soft Constraints*, 2004.
11. Anthony Wren. Scheduling, timetabling and rostering – a special relationship? In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 46–75. Springer-Verlag LNCS 1153, 1996.

Lecture and Tutorial Timetabling at a Tunisian University

Abdelaziz Dammak, Abdelkarim Elloumi, Hichem Kamoun

University of Sfax, FSEG, Route de l'Aerodrome, km 4, BP 1088, 3018 Sfax Tunisia

Department of Applied Quantitative Methods, GIAD Laboratory

{abdelaziz.dammak, abdelkarim.elloumi,
hichem.kamoun}@fsegs.rnu.tn

1 Introduction

This paper deals with the Lecture and Tutorial Timetabling Problem at an institution in a Tunisian University. Our objective is to construct a feasible timetable for all lectures and tutorials taken by different groups of each subsection of any section in the institution. For this, we describe the timetabling problem of the institution considered and list all specific hard and soft constraints. We formulate the problem as a zero-one integer linear program in which we define two binary variables corresponding respectively to lectures and tutorials. The quadratic objective function proposed tries to eliminate a real problem of congestion leading to a waste of time and students' delays. Since the number of constraints is very large, the use of the heuristic procedures is of primary importance. We develop three heuristic procedures: first, we start by assigning all lectures of different student sections having the biggest size in a classroom with the smallest capacity that can fit the students. Second, we complete the output of the first phase by assigning the tutorials for different groups. Lectures and tutorials timetabling problem are correlated and cannot be treated independently if we desire to get a complete solution. The two first heuristics are illustrated with real data of one section at the Faculty of Economics and Management Sciences of Sfax in Tunisia and compared with those manually generated. Since there are several criteria which are preferably satisfied as much as possible, we will formulate the problem as a multiobjective mathematical program, and then we develop later a third heuristic in order to ameliorate the quality of the solution. The different criteria which can be taken into account are: minimize the number of free inter-meetings, maximize the professor preferences, minimize the distance covered by the students between the classrooms and exempt the students as much as possible in the half day.

Educational timetabling has been the subject of several papers in various scientific journals and the topic of many theses in different universities. This problem concerns essentially course and exam which are to be scheduled during the academic year. The course timetabling problem consists of scheduling a certain number of courses into a certain number of timeslots spread throughout the week in such away that hard and soft constraints are satisfied.

Various techniques have been used to solve Timetabling Problems (see Burke et al. [3], Carter and Laporte [8]). One of the earliest methods used to solve this problem is graph colouring in which vertices represent events and two vertices are connected if

and only if there is a conflict. Welsh and Powell [18], Wood [19], Selim [16] and De Werra [12] proposed several formulations by graph colouring for a set of class-teacher timetabling problems and discussed the inherent complexity. Recently, Timothy [17] has used graph colouring to solve both course and exam timetabling.

Linear and integer programming models were frequently used to formulate the course time-tabling problem usually with binary variables (Diskalaki et al. [11], Diskalaki et Birbas [10] and Dimopoulos and Miliotis [13], [14]).

Burke and Petrovic [2] discuss some recent development in the field of automated timetabling. The discussion deals with both course and exam timetabling. Overviews of four types of approaches to timetabling problems that have been applied are given: sequential methods, cluster methods, constraint based methods and metaheuristic methods.

Another technique that has recently been successfully applied to course timetabling is Case-Based Reasoning (CBR). The origin of CBR dates back to 1977 with the work of Schank and Abelson [15]. CBR has also been well applied to scheduling and optimization problems.

Burke et al. ([1], [4], [5], [6]) were the first to adapt this approach to solve university timetabling problems. The main idea behind the use of CBR in timetabling is that previous timetabling problems and their appropriate solution procedures are stored in a knowledge base which is used to provide good solution for a new timetabling problem.

In the papers [4] and [6], the authors illustrate the use of attribute graphs to graphically represent a course timetabling problem. In this graph, the courses (events) are represented by nodes and the relationship that exists between these events (including hard and soft constraints) is indicated by edges. Then a similarity measure is used to indicate which part of the attribute graphs of the stored cases in the knowledge base has the most similar structure of the attribute graph of a new timetabling problem. Finally, the most appropriate solution procedure used for the selected stored case is adapted to solve the new problem.

In the paper [1], they keep using case-based reasoning approach for solving course timetabling problem but instead of using attribute graphs for constructing the knowledge base, a knowledge discovery process is performed based on a set of features that are judged to be most appropriate to describe the characteristics of the timetabling problem.

In the recent paper [5], they use the multiple-retrieved case-based reasoning approach to solve large scale timetabling problem which were until then unable to be solved by CBR in the earlier papers. The main idea is to decompose the attribute graph associated with the large timetabling problem into smaller attribute sub-graphs whose associated timetabling problem can be solved using CBR approach. Then the partial solutions are all combined to obtain a timetable for the large time-tabling problem.

In their article in press, Burke et al. [7] develop a graph-based hyper-heuristic (GHH) which has its own search space that operates in high level with the solution space of the problem generated by the so-called low level heuristics.

2 Problem description

The construction of course timetabling at the Faculty of Economics and Management Sciences of Sfax (FEMSS) is performed manually by administration staff twice in each academic year (first and second semester). There are thirty timeslots distributed along the six days of the week: Monday to Saturday. There are six timeslots in Monday, Tuesday, Thursday and Friday and only three timeslots in the morning in Wednesday and Saturday. Each timeslot has one hour and a half duration followed by fifteen minutes break except the third timeslot in the morning is followed by thirty minutes lunch break.

The lectures and tutorials are of two categories: there are some with only one period per week and others require two periods per week. Lectures with two periods cannot be held at the same day. There are lectures without tutorial, with only one period tutorial and with two-period tutorial. There are several sections divided into different subsections.

Each subsection with a big size is divided into a certain number of groups having a size no more than thirty students. The lectures are to be taught to a whole section or subsection while the tutorials are only taught to groups in small classrooms.

As any timetabling problem, there are both hard and soft constraints. The hard constraints are those that cannot be violated at any circumstances in order to obtain a feasible solution.

We consider these hard constraints:

- All courses (lectures and tutorials) included in the program of each section are insured.
- Any professor cannot teach more than one course at the same period.
- Any classroom cannot be used more than once in any period.
- Any group of any subsection of any section cannot be taught more than one course in any timeslot.
- Any subsection of any section cannot take two lectures in two consecutive timeslots.
- Courses with two periods cannot be taught twice in the same day.
- Any professor does not teach three courses at three consecutive timeslots.
- Any group of any subsection of any section cannot be taught consequently in the third and fourth periods.
- Any professor cannot teach consequently in the third and fourth periods.

The soft constraints are restricted to:

- The time preferences of professors should be respected as much as possible.
- For any group of any section the rate of occupation of the seats should be maximized.

Other soft constraints can be considered:

- Minimize the number of free inter-meetings.
- Maximize the professor preferences.
- Exempt the student as far as possible in the half day.

3 Problem formulation

The course timetabling problem of the Tunisian institution was formulated as a zero-one linear integer program in which we define two binary variables corresponding respectively to lectures and tutorials. In this formulation, we have considered all hard constraints cited in section 2.

The objective function has a quadratic form in which we have considered a real problem of routing between classrooms and aims to minimize the distance covered by students between these classrooms.

4 Tutorials' Timetabling Heuristic (TTH)

This heuristic completes the one that has been developed by Damnak et al. [9] in which the authors solve the problem of lecture timetabling in the same institution. This new heuristic is composed of eight steps detailed as follow:

Step (1):

Arrange the set of sections in non-increasing order of the enrolled student size.

Arrange the classrooms in non-increasing order of their size.

Step (2):

For each group of each subsection of each section, we begin by assigning the first period of the two-period tutorial that needs to be taught to this group of the subsection.

Step (2.1):

We look for the first classroom which can hold the current tutorial and having the smallest size.

Step (2.2):

This classroom is assigned to this tutorial if and only if:

We find a period in which this classroom is available and at the same time the group of the subsection is free in the current period. If this current period is the third (respectively the fourth) in the day then the group has to be free the fourth (respectively in the third) period.

In case no such period exists, we check the availability of the immediately precedent classroom.

If there is no classroom available that can fit this tutorial, we have to divide the group into smaller groups.

Also we consider the availability and time preferences of professors that can teach this tutorial.

Step (2.3):

For a certain period, we check if the professor has taught in the two consecutive preceding periods or in the two consecutive following periods or in the two periods corresponding to the previous and the following periods.

Step (2.4):

If the current professor is busy or step (2.3) is satisfied then choose another professor.

Step (3):

We assign the one-period tutorials. We follow the same procedure used in step (2.1) to step (2.4) (respectively) denoted step (3.1) to step (3.4) (respectively).

Step (4):

We assign the second-period of the two-period tutorials that has to be taught by the same professor. We follow the same procedure of steps (2.1) and (2.2) (respectively) denoted steps (4.1) and (4.2) (respectively). In addition, we have to prevent the assignment of the second period tutorial during the same day in which the first period tutorial is scheduled.

5 Numerical Example

We restrict our numerical example on only one group chosen from the first subsection of the first section of the institution. We denote C_{ijk} the lecture k taught by the subsection j of section i , D_{ijt} the tutorial t taught by the subsection j of section i , s the classroom, and h the professor. The output of our heuristic is summarized in the timetable 1:

Timetable 1

Day / Hour	08 – 9:30	09:45-11:15	11:30-13:00	13:30-15:00	15:15-16:45	17:00-18:30
Monday	$C_{111}, s = 3,$ h=1	$D_{112}, s = 28,$ h=17		$C_{113}, s = 3,$ h=5	$D_{116}, s = 28,$ h=29	$D_{114}, s = 28,$ h=6
Tuesday	$C_{112}, s = 3,$ h=2	$D_{118}, s = 28,$ h=36	$C_{114}, s = 3,$ h=6		$C_{115}, s = 3, h=7$	$D_{117}, s = 28,$ h=37
Wednesday	$C_{111}, s = 3,$ h=1	$D_{111}, s = 28,$ h=14				
Thursday	$C_{112}, s = 3,$ h=2	$D_{114}, s = 27, h=6$		$C_{113}, s = 3,$ h=5		$D_{113}, s = 24,$ h=20
Friday	$C_{116}, s = 3,$ h=9					
Saturday						

Timetable 2

Day / Hour	08 – 9:30	09:45-11:15	11:30-13:00	13:30-15:00	15:15-16:45	17:00-18:30
Monday			$D_{112}, s = 31,$ h=2	$C_{115}, s = 2,$ h=7	$D_{114}, s = 46, h=6$	
Tuesday	$D_{118}, s = 35,$ h=36			$C_{116}, s = 3,$ h=8	$C_{113}, s = 2, h=5$	
Wednesday	$C_{114}, s = 2,$ h=6	$C_{111}, s = 1,$ h=1				
Thursday	$C_{112}, s = 4,$ h=2	$D_{114}, s = 44, h=6$		$D_{117}, s = 36,$ h=33		
Friday	$C_{112}, s = 3,$ h=2	$C_{111}, s = 1,$ h=1	$D_{116}, s = 58,$ h=28		$C_{113}, s = 3, h=5$	$D_{113}, s = 43,$ h=19
Saturday	$D_{111}, s = 52,$ h=12					

To illustrate the performance of our heuristic, we compare the results presented in the timetable one with those generated manually by the administration presented in the timetable two.

From these two timetables, we can draw the following remarks:

1. In the manual solution, the number of half days with single lesson is equal to 4. However, in the solution provided by the heuristic (TTH), there is only one half day with one meeting. It is preferable to schedule at least two courses per half day in order to prevent the student moving for only one meeting.
2. The advantage of the output of our heuristic consists of releasing students as far as possible during the weekend (morning of Friday). For this, the amelioration of the solution is easier to perform in the heuristic solution than in the manual one.
3. The constraint of excluding third and fourth timeslots in each of the four completes days is rigorously satisfied by heuristic solution but not considered by hand-made one.
4. Finally, this comparison is far from being definitive and conclusive since this work is considered partial for the following reasons. First, we considered only one group of one subsection of one section; a thorough test will include all sections. Second, data for at least three recent academic years need to be used in the test.

References

1. Burke E. K, MacCathy, B. Petrovic, S. and Qu, R. knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning. In Burke, E.K. and De Causmaeker, P. (editors), PATAT'02, LNCS 2740 (2002) 90-103
2. Burke, E.K. and Petrovic, S. Recent research directions in automated timetabling. European Journal of Operations Research, 14(2) (2002) 266-280
3. Burke, E.K., Jackson, K., Kingston, J. and Weare, R. automated university timetabling: the state of the art. The computer journal, 40(9) (1997) 565-571
4. Burke, E.K., MacCathy, B., Petrovic, S. and Qu, R. case-based reasoning in course timetabling: an attribute graph approach, In Proceedings of 4th International Conference on Case-Based reasoning (ICCBR 2001). LNAI 2080 (2001) 90-105
5. Burke, E.K., MacCathy, B., Petrovic, S. and Qu, R. multiple-retrieval case-based reasoning for course timetabling problems. Journal of the Operational Research Society (2005) 1-15
6. Burke, E.K., MacCathy, B., Petrovic, S. and Qu, R. structured case in case-based reasoning-re-using and adapting cases for timetabling problems. Knowledge-Based Systems. 13 (2000) 159-165
7. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S. and Qu, R. A graph-based hyper-heuristic for educational timetabling problem. European Journal of Operational Research, article in press (2006) 1-16
8. Carter, M.W. and Laporte, G. recent developments in practical course timetabling, In Burke, E. K. and Carter M. W. (editors) PATAT'97, LNCS 1408 (1997) 3-19
9. Dammak, A., Elloumi, A. and Kamoun, H. Lecture timetabling at a Tunisian university. Technical report No 4, GIAD Laboratory (2006) 1-25
10. Daskalaki, S. and Birbas, T. Efficient solutions for university timetabling problem through integer programming. European Journal of Operational Research. 160 (2005) 106-120

11. Daskalaki, S., Birbas, T. and Housos, E. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*. 153 (2004) 117-135
12. De Werra D. Some combinatorial models for course scheduling, In Burke, E. K. and Ross, P. (editors) PATAT'95, LNCS 1408 (1995) 296-308
13. Dimopoulou, M. and Miliotis, P. An Automated Course Timetabling System developed in a distributed Environment: a Case Study. *European Journal of Operational Research*. 153 (2004) 136-147
14. Dimopoulou, M. and Miliotis, P. Implementation of a University Course and Examination Timetabling System. *European Journal of Operational Research*, 130 (2001) 202-213
15. Schank, R.C. and Abelson, R.P. Scripts, plans, goals and understanding. Erlbaum, Hillsdale, New Jersey, US (1977)
16. Selim, S.M. Split Vertices in Vertex colouring and their application in developing a solution to the faculty timetable problem. *The Computer Journal*, 31(1) (1988) 76-82
17. Timothy, A.R. A Study of university timetabling that blends graph coloring with the satisfaction of various essential and preferential conditions, Ph.D. Thesis. Rice University (2004)
18. Welsh, D. J. A. and Powell, M. B. An upper bound for the chromatic number of graph and its application to timetabling problems. *The Computer Journal*, 10 (1) (1967) 360-364
19. Wood, D. C. A technique for colouring a graph applicable to large scale timetabling problems. *The Computer Journal*, 12 (4) (1969) 317-319

An employee timetabling problem in a maintenance service of a software company

Laure-Emmanuelle Drezet¹, Deborah Chesnes² and Odile Bellenguez-Morineau²

¹ Ecole des Mines de Saint-Etienne, CMP Georges Charpak, Avenue des Anemones - Quartier Saint-Pierre, 13541 Gardanne, France
drezet@emse.fr

² Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France
odile.morineau@univ-tours.fr; chesnes@etu.univ-tours.fr

1 Introduction

The considered problem occurs in the maintenance service of a French software company, Delta Informatique. This company sells software that allow to manage bank information systems. The 24 employees of the maintenance service have to ensure the maintenance of these software. They have also to solve dysfunctions that could occur in these products or to give assistance to customers. The employees' timetables have though to satisfy the clients' requests, within a planning horizon of three months.

The help desk is closed only one day per week (from Saturday at 2:00PM to Sunday at 2:00PM). Four shifts are defined for the help desk: MS - Morning shift (from 6:00 AM to 2:00 PM), AS - Afternoon shift (from 2:00 PM to 10:00 PM), NS - Night shift (from 10:00 PM to 6:00 AM), SS - Sunday shift (from 2:00 PM to 0:00 AM). A shift is defined for the employees not assigned to the help desk: DS - Day Shift (from 9:00 AM to 6:00 PM, with a noon break). To respond to clients requests, two employees have to be assigned to MS, AS and to NS, from Monday to Friday. On Saturday, two employees are assigned to MS. On Sunday, two employees are assigned to SS. Employees not assigned to help desk, work from Monday to Friday on a classical shift, DS. These employees develop new functions in software.

In this problem, several hard constraints have to be respected. Obviously, the workforce requirements and the labour legislation must be respected. Each employee must have at least eleven hours between two working days. Each employee must have at least two days-off per week. Moreover, if an employee works on week-ends or on public holiday, some days-off have to be inserted in his/hers timetable.

Another constraint concerns the spread of skills among the present employees. The employees of the maintenance service are divided in five different skill

groups. Each skill group masters a specific part of the software. When an employee is assigned to the help desk, her/his skill group has one unavailable employee. Though the employees assigned to the help desk must belong to different skill groups. Moreover, the customers could call the help desk for different problems. Two employees that are assigned to the same shift in the help desk have to belong to different skill groups, such that the number of mastered skills at the help desk is maximised.

Employees have also different levels: junior or senior. A senior employee has been working for the company for a long time and masters more skills than a junior employee. As only two employees work at night, at least one of them has to be a senior employee.

Several preferences of employees have also to be considered. Each employee can specify a few number of days when she/he can not be assigned to specific shifts. Each employee have also to take a given number of days-off within the three months. A part of these days are fixed by the employee. The remaining days are set by the planner, with respect to hard constraints.

In addition to all these hard constraints, timetables should be as fair as possible. This fairness is evaluated by computing, for each employee, the number of worked nights, week-ends or public holidays and the number of satisfied preferences. Car sharing could also be considered, such that two employees can specify that they prefer to work on the same shifts.

We can observe that this problem is quite close to Nurse Scheduling Problem (as in [1]). As the planner builds timetables by hand, our goal is to propose an automated procedure that allows to find feasible solution quite rapidly.

2 Model

First of all, we have developed an Integer Linear Program. The model is very huge, as all the hard and the soft constraints are considered. The objective function (to minimize) is a weighted sum of all the violated soft constraints and a measure of unfairness. This approach was developed to know the limits of this type of exact approach for this type of problem. Some computational experiments have been carried. As forecasted, with a free solver (GLPK), it takes more than a day to compute a simple timetable for about fifteen employees, for two weeks. As mentioned in [2], it is not surprising. In fact, Integer Linear Programming is not very efficient when global constraints or employees' preferences have to be considered.

Then a priority-rule based heuristic was developed. This greedy algorithm reproduces the human way to build timetables and its first results are encouraging (Timetables are computed in less than five minutes for all the employees

and for three months). This method uses different priority rules and different strategies to assign shifts to employees.

3 Conclusion

The studied timetabling problem is quite different from classical timetabling problems, due to some specific hard constraints and to the huge number of considered soft constraints. Mathematical programming could not be used for solving this problem, due to the size of the model. The greedy method we propose obtain good-quality solutions, in less than five minutes. These results can be improved by using a metaheuristic, such as a genetic algorithm.

References

1. Bellanti, F., Carello, G., Della Croce, F., Tadei, R.: A greedy-based neighborhood search approach to a nurse rostering problem In: European Journal of Operational Research (2004), Vol. 153, 28–40
2. Meisels, A., Gudes, E., Solotorevsky, G.: Combining rules and constraints for employee timetabling In: International Journal of Intelligent System (1997), Vol. 12, 419–439

Referee Assignment in Sports Tournaments

Alexandre R. Duarte¹, Celso C. Ribeiro², and Sebastián Urrutia¹

¹ Department of Computer Science, Catholic University of Rio de Janeiro,
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil.

² Department of Computer Science, Universidade Federal Fluminense,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.

{aduarte,celso,useba}@inf.puc-rio.br

1 Introduction

Optimization in sports is a field of increasing interest. Some applications have been reviewed by Ribeiro and Urrutia [7]. Combinatorial optimization techniques have been applied e.g. to the traveling tournament problem [3, 9], to playoff elimination in championships [8], and to the scheduling of a college basketball conference [6]. Easton et al. [2] reviewed scheduling problems in sports.

A common problem found in sports management is the assignment of referees to games already scheduled. The number of referees to be assigned to each game may vary depending on the sport or the league. For example, soccer games usually require three referees, while basketball games require only two. 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 in amateur leagues, 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 also players (or players' relatives). In this case, a natural constraint is that a referee cannot officiate a game that he/she is scheduled to play.

Referee assignment problems in other contexts have been addressed in [4, 5, 10]. Dinitz and Stinson [1] considered a problem involving referee assignment to tournament schedules, connecting room squares and balanced tournament designs. In this work, we address a simplified 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 describe the main constraints and the objective function of the problem. The proposed solution strategy is summarized in Section 3. 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 time slots where they take place are known beforehand. In our approach, referees are assigned to empty refereeing positions, not to games. This approach allows not only to handle referee assignment problems in different sports, but also problems in tournaments where different games may need different numbers of referees. 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 a *referee slot*.

Each referee slot has to be filled by a referee with a minimum skill level, which is previously determined and often related to the tournament division. Usually, a division corresponds to a set of teams formed by players under a certain age and with the same gender, e.g. boys under 16 years old. Each referee has a certain skill level defining the games he/she can officiate. Additionally, referees may declare their unavailability to officiate at certain time slots. Furthermore, each referee establishes the maximum number of games he/she is able to officiate and 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.

The Referee Assignment Problem (RAP) consists in assigning referees to all referee slots associated to games scheduled to 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:

- all referee slots must be filled for all games;
- referees cannot officiate games in referee slots overlapping time slots where they are already scheduled to play or to officiate;
- referees cannot officiate games in referee slots where they declared to be unavailable;
- referees must meet the minimum skill level established for each referee slot;
- referees cannot officiate more than a given maximum number of games; and
- referees cannot officiate in two or more different facilities on the same day.

3 Solution strategy

The problem described in the previous section was formulated by integer programming. Only small instances with up to 40 games and 40 referees could be exactly solved by a commercial solver such as CPLEX 9.0.

We propose a three-phase heuristic approach to tackle real-life large instances of the referee assignment problem:

1. Apply a greedy heuristic to find an initial solution, possibly violating some constraints.
2. Make the initial solution feasible, by using a local search repair heuristic based on swap moves (referees assigned to two referee slots are swapped), exchange moves (the referee assigned to a referee slot is replaced by another referee), and group perturbations (all referee slots assigned to two referees officiating at different facilities are swapped).

3. Improve the feasible solution built in the previous step, by using a local search procedure based on exchange moves, simple perturbations (swap moves that do not change the value of the objective function), and group perturbations.

Randomly generated instances following patterns similar to real-life applications have been used in the computational experiments. They have up to 500 games and up to 1250 referees, different numbers of facilities, and different patterns of the target number of games each referee is willing to officiate. Preliminary experiments on a standard Pentium IV processor have shown that the greedy heuristic was able to build feasible solutions for the largest instances in less than 0.1 second. When a feasible solution was not found after the first phase, the local search repair heuristic took less than 1.0 second to find one. The third phase typically improved the feasible initial solutions by 50%.

4 Extensions

We are currently working on extensions addressing further constraints of real-life applications. One of such extensions is the existence of hard or soft links between some referees. In this case, 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. In some situations, some referees can be unable to travel. Moreover, managers may want 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 into 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 and we are also addressing the use of multi-criteria methods coupled with decision support systems for its solution in practice.

References

1. J.H. Dinitz and D.R. Stinson. On assigning referees to tournament schedules. *Bulletin of the Institute of Combinatorics and its Applications*, 44:22–28, 2005.
2. K. Easton, G.L. Nemhauser, and M. Trick. Sports scheduling. In J.T. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 52.1–52.19. CRC Press, 2004.
3. K. Easton, G.L. Nemhauser, and M.A. Trick. The traveling tournament problem: Description and benchmarks, principal and practices of constrained programming. *Lecture Notes in Computer Science*, 2239:580–585, 2001.

4. J.R. Evans. A microcomputer-based decision support system for scheduling umpires in the American baseball league. *Interfaces*, 18:42–51, 1988.
5. J.R. Evans, J.E. Hebert, and R.F. Deckro. Play ball - the scheduling of sports officials. *Perspectives in Computing*, 4:18–29, 1984.
6. G.L. Nemhauser and M.A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1997.
7. C.C. Ribeiro and S. Urrutia. OR on the ball: Applications in sports scheduling and management. *OR/MS Today*, 31:50–54, 2004.
8. C.C. Ribeiro and S. Urrutia. An application of integer programming to playoff elimination in football championships. *International Transactions in Operational Research*, 12:375–386, 2005.
9. C.C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, to appear.
10. M.B. Wright. Scheduling English cricket umpires. *Journal of the Operational Research Society*, 42:447–452, 1991.

Branch-and-cut for a real-life highly constrained soccer tournament scheduling problem

Guillermo Durán¹, Thiago F. Noronha², Celso C. Ribeiro³, Sebastián Souyris¹,
and Andrés Weintraub¹

¹ Department of Industrial Engineering, University of Chile, Republica 701, Chile.

² Department of Computer Science, Catholic University of Rio de Janeiro,
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil.

³ Department of Computer Science, Universidade Federal Fluminense,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.

{tfn,celso}@inf.puc-rio.br, {gduran, ssouyris, aweintra}@dii.uchile.cl

1 Introduction

There are 20 teams in the Chilean soccer first division. They take part in two yearly tournaments: *opening* and *closing*. Each tournament is organized in two phases: *qualifying* and *playoffs*. The qualifying phase follows the structure of a single round robin tournament. 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. The first four teams are distributed on the four groups. The teams from 5th to 8th place are randomly distributed in different groups. This scheme 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 National Association of Professional Football (ANFP) is in charge of scheduling the games of the opening and closing tournaments. Good schedules are of major importance for the success of the tournaments, making them more balanced, profitable, and attractive. All schedules were randomly prepared until 2004. Weintraub et al [1] addressed the main drawbacks of such schedules and tackled the problem by integer programming. Their model is applied since 2005. However, the computation times are very high and the solutions produced by the model still lack quality.

In this work, we improve the original integer programming formulation. Valid inequalities are derived and appended to the model. A new branch-and-cut strategy is used to speedup convergence. The main constraints and the objective function are described in Section 2. The solution approach and the branching strategy are summarized in Section 3. Preliminary results on a real-life instance are reported in the last section.

2 Problem statement

A HAP stands for a home-away pattern and defines a sequence of home and away games for a given team. *Popular* teams are those with more fans. *Traditional*

teams are the oldest teams. *Strong* teams are those better qualified in the last tournaments. *Tourist* teams are those from Viña del Mar, Valparaíso, and La Serena. A *classic* is a game between two traditional teams. The constraints of the problem are the following:

- Each team plays against every other team exactly once.
- Every team plays exactly once in each round.
- Each team plays at least nine games at home and nine games away.
- A team may never have two consecutive breaks [5].
- A team may play at most three games at home in any five consecutive rounds.
- Some teams have complementary HAPs (whenever one of them plays at home the other plays away, and vice-versa).
- There may be at most four games in Santiago in any round.
- There may be no breaks in rounds 1, 16, and 18.
- Pairs of excluding teams: if a team plays against one of them at home, then it should play away against the other (and vice-versa).
- Classics should not be played before round 7 or after round 16.
- No team can play two consecutive games against popular teams.
- No team can play two consecutive games against strong teams.
- Each traditional team plays exactly one classic at home.
- Tourist teams should play at least once against a traditional team during the summer rounds.
- Traditional teams cannot play twice in the same week in the same tourist region.
- A team from the Central region cannot play in the same week against a team from the South and another from the North.

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 be held in the last rounds. The objective function consists in maximizing the number of games between teams in the same group in the last rounds of the tournament. It is given by the sum of the indices of the rounds where the games between teams of the same group take place.

3 Solution approach and branching strategy

The problem is formulated by integer programming and solved by a branch-and-cut algorithm. Two variables were used in the original model [1]: $x_{ijk} = 1$ if team i plays at home against team j in round k , $x_{ijk} = 0$ otherwise; $y_{ik} = 1$ if team i has a break at round $k + 1$ (i.e., team i plays two consecutive home games or two consecutive away games in rounds k and $k + 1$), $y_{ik} = 0$ otherwise.

The new formulation follows the same strategy proposed by Trick [3] and is based on the introduction of a new binary variable: $z_{ik} = 1$ if team i plays at home in round k , $z_{ik} = 0$ otherwise. All HAP constraints are rewritten in terms of this new variable. This formulation is more easily solvable. Furthermore, the new variable plays a major role in the branching strategy. Each round is a perfect

matching in the complete graph whose nodes are the participating teams [2, 4]. Cuts associated with the most violated matching constraints in the linear relaxation are progressively added to the enumeration tree.

The branching strategy plays a major role in the success of a branch-and-cut algorithm. Branching on the x_{ijk} variables is not efficient, since most of them are null in integral solutions. Our branching strategy is based on the z_{ik} variables. Branching on the x_{ijk} variables starts only after all the z_{ik} variables are integral. This strategy implicitly decomposes the solution in two phases. In the first phase the HAPs for each team are computed, while in the second the dates of the games are established. Once the variables z_{ik} are fixed, the branch-and-cut algorithm needs just a few branches on variables x_{ijk} to find a feasible solution.

4 Preliminary results

Two algorithms based on the previous formulation have been proposed and evaluated: the B&C-ANFP branch-and-cut algorithm and the B&B-ANFP branch-and-bound algorithm without cuts. Both of them were implemented using the library Concert Technology 1.2 and version 8.0 of the CPLEX solver. 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 opening tournament, comparing them with those reported in [1].

Computation times for solving the linear programming relaxation by different algorithms available with the CPLEX 8.0 package are given in Table 1. The problem is very degenerated and requires the use of perturbations, leading to large computation times. The interior point algorithm was the best solution strategy. We also can see in Table 1 that the new formulation considerably reduced the computation time of the interior points algorithm, making possible an efficient implementation of the cutting plane algorithm.

Table 1. Computation times for solving the linear relaxation.

Strategy	time (s)
Primal simplex	27
Dual simplex	21
Interior points (original formulation)	12
Interior points (variables z_{ik})	4

Results obtained with algorithms B&B-ANFP and B&C-ANFP are given in Table 2, which reports the value of the objective function, the number of nodes in the enumeration tree, and the integrality gap after some elapsed time. In the beginning, algorithm B&B-ANFP finds good solutions faster than B&C-ANFP. However, the former was not able to find the optimal solution after 4 hours. 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 optimal solution in less than two hours of computation time.

Table 2. 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 minutes	617	140	3.6	474	120	25.9
30 minutes	622	600	2.9	615	330	3.9
1 hour	631	1120	1.4	633	570	1.1
2 hours	633	2190	1.1	640	1560	0.0
4 hours	639	4860	0.2	—	—	—

In Table 3, we compare the results obtained by algorithm **B&C-ANFP** with those obtained by the approach in [1]. We give the value of the objective function and the integrality gap after 30 minutes and after two hours of computation time (on a 2.4 GHz Pentium IV computer for [1]) for both algorithms. Algorithm **B&C-ANFP** not only found a better solution (615) very quickly (30 minutes), but also found the optimal solution value (640) after the same time that the approach in [1] took to find a solution value 10.0% away from the optimal.

Table 3. Comparison between algorithms **B&C-ANFP** and Weintraub et al [1].

Algorithm	time	objective	gap (%)
B&C-ANFP	30 minutes	615	3.9
	2 hours	640	0.0
Weintraub et al [1]	2 hours	576	10.0

References

1. G. Durán, M. Guajardo, J. Miranda, D. Sauré, S. Souyris, A. Weintraub, A. Carmash, and F. Chaigneau. Programación matemática aplicada al fixture de la primera divisón del fútbol chileno. *Revista Ingeniería de Sistemas*, 5:29–46, 2005.
2. M.W. Padberg and M.R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operation Research*, 7:67–80, 1982.
3. M.A. Trick. A schedule-and-break approach to sports scheduling. *Lecture Notes in Computer Science*, 2079:242–253, 2001.
4. M.A. Trick. Integer and constraint programming approaches for round robin tournament scheduling. *Lecture Notes in Computer Science*, 2740:63–77, 2003.
5. S. Urrutia and C.C. Ribeiro. Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. *Discrete Applied Mathematics*, to appear.

Constructive Algorithms for the Constant Distance Traveling Tournament Problem

Nobutomo Fujiwara¹, Shinji Imahori¹,
Tomomi Matsui², and Ryuhei Miyashiro³

¹ 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

² Department of Information and System Engineering,
Faculty of Science and Engineering, Chuo University,
Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan

matsui@ise.chuo-u.ac.jp

³ 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

1 Constant Distance Traveling Tournament Problem

In this abstract, we deal with the Constant Distance Traveling Tournament Problem (CDTTP) [4], which is a special class of the Traveling Tournament Problem (TTP), established by Easton, Nemhauser and Trick [1]. We propose a lower bound of the optimal value of CDTTP, and two algorithms that produce feasible solutions whose objective values are close to the proposed lower bound. For some size of instances, our algorithms yield feasible solutions better than the previous best solutions.

In the following, several definitions for CDTTP are introduced. Given even n , the number of teams, a double round-robin tournament is a set of games in which every team plays every other team exactly once at home and once at away. A game is specified by an ordered pair of opponents. Exactly $2(n-1)$ slots or time periods are required to play a double round-robin tournament. Each team begins at its home site and travels to play its games at the chosen venues. Each team then returns (if necessary) to its home at the end of the schedule. The number of trips of a team is defined by the number of moves of the team between team sites. Consecutive away games for a team constitute a road trip; consecutive home games are a home stand. The length of a road trip or home stand is the number of opponents playing against in the road trip/home stand. The problem CDTTP is defined as follows.

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 is 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.

The CDTTP and its variations are discussed in [3, 5]. The CDTTP can be considered as a special class of the original TTP [1] such that all distances between team sites are one.

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

2 Lower Bound

We proved the following lemma that provides a lower bound of the optimal value of CDTTP. Due to space limitation, the proof is omitted.

Lemma 1. *The total number of trips of every feasible schedule of n teams is greater than or equal to $\text{LB}(n)$ defined by*

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

3 Algorithms

We propose two algorithms for constructing feasible schedules by modifying single round-robin tournaments. Due to space limitation, for both algorithms we describe these procedures for the case of $n \equiv 1 \pmod{3}$, and show only the results for $n \in \{0, 2\} \pmod{3}$.

3.1 Modified Circle Method

First, we propose the algorithm named *Modified Circle Method* (MCM). 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

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

For any permutation π on T , $G^e(\pi)$ denotes the set of $n/2$ matches satisfying that every directed edge $(u, v) \in A^e$ corresponds to a match between $\pi(u)$ and $\pi(v)$ held at the home of $\pi(v)$. For each $j \in \{1, 2, \dots, n-1\} = T \setminus \{n\}$, we define a permutation π^j by $(\pi^j(1), \pi^j(2), \dots, \pi^j(n)) = (j, j+1, \dots, n-1, 1, \dots, j-1, n)$. Let G^o be a directed graph obtained from G^e by reversing the direction of the edge between 1 and n . Let X be a single round-robin tournament satisfying that matches in slot s are defined by $G^o(\pi^s)$ (if $s \in \{1, 2, 3\} \pmod{6}$) and $G^e(\pi^s)$ (if $s \in \{4, 5, 0\} \pmod{6}$). 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 . Now

we construct a feasible schedule Y by concatenating partial schedules consisting of three slots as follows:

$Y = (X_1, \overline{X_1}, \overline{X_2}, X_2, X_3, \overline{X_3}, \overline{X_4}, X_4, X_5, \dots, X_{\frac{n-1}{3}}, \overline{X_{\frac{n-1}{3}}})$, where $\overline{X_i}$ is a partial schedule obtained from X_i by reversing venues.

For $n \equiv 1 \pmod{3}$, MCM produces a feasible schedule Y of which the total number of trips is $(4/3)n^2 - (1/2)n - 4/3 = \text{LB}(n) + (1/3)n - 1/3$. Since we do not have the space for describing MCM for $n \in \{0, 2\} \pmod{3}$, we simply state our results as below. Let the total number of trips of a feasible schedule Y be $w(Y)$.

Theorem 1. *The Modified Circle Method produces a feasible schedule Y such that*

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

3.2 Minimum Break Method

Here we propose the algorithm named *Minimum Break Method* (MBM). The procedure of MBM is also described only for the case of $n \equiv 1 \pmod{3}$.

Given a feasible schedule, it is said that a team has a *break* at slot s if it has two consecutive home games (home break) or two consecutive away games (away break) in slots $s-1$ and s . The total number of breaks $b(Y)$ is defined as the sum of the number of breaks of all the teams in a feasible schedule Y .

Let X be a schedule of a single round-robin tournament satisfying the following conditions:

- (C1) the number of breaks $b(X)$ is equal to $n-2$;
- (C2) at each slot $s \in \{3, 5, 0\} \pmod{6}$, exactly two teams have a break.

When $n \leq 50$, we have obtained a single round-robin tournament satisfying (C1) and (C2) by solving integer programming problems (e.g., see [2]). Now we construct a single round-robin tournament X' from X by reversing venues for each even slot. Then X' satisfies that exactly two teams have $n-2$ breaks, other teams have $n-3$ breaks, and every team has a break at slot s satisfying $s > 1$ and $s \equiv 1 \pmod{3}$. 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 . Now we construct a feasible schedule Y' by concatenating partial schedules consisting of three slots as follows:

$Y' = (X'_1, \overline{X'_1}, X'_2, \overline{X'_2}, X'_3, \overline{X'_3}, X'_4, \overline{X'_4}, X'_5, \dots, \overline{X'_{\frac{n-1}{3}}})$, where $\overline{X'_i}$ is a partial schedule obtained from X'_i by reversing venues.

For $n \equiv 1 \pmod{3}$, the above procedure produces a feasible schedule Y' such that $w(Y') = (4/3)n^2 - (5/6)n - 1 = \text{LB}(n)$. For $n \in \{0, 2\} \pmod{3}$, again we simply state our results as follows.

Table 1. Results for $16 \leq n \leq 24$

n	LB(n)	MCM	MBM	known
16	327	332	*327	327
18	414	419	426	418
20	520	535	529	521
22	626	633	*626	632
24	744	751	755	757

*: our solutions that attain the lower bound LB(n)

known: the known best solutions in [4], as of April 2006

Theorem 2. *If there is a round-robin tournament satisfying Conditions (C1) and (C2), the Minimum Break Method produces a feasible schedule Y' such that*

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

As mentioned before, we have already obtained schedules satisfying (C1) and (C2) for $n \leq 50$. Using MBM with them as initial solutions, we obtained feasible schedules (see Table 1).

Lastly, we summarize our results. For $n \equiv 0 \pmod{3}$, MCM gives better solutions compared to MBM. In contrast, for $n \in \{1, 2\} \pmod{3}$ MBM performs better though it needs an initial schedule satisfying Constraints (C1) and (C2). In addition, when $n \equiv 1 \pmod{3}$, with an initial schedule MBM yields a solution that attains LB(n), i.e., an optimal solution. Table 1 shows the results for $16 \leq n \leq 24$: for $n = 24$ both algorithms produced better solutions than the previous best; for $n = 22$ MBM gave an optimal solution.

References

1. Easton, K., Nemhauser, G., Trick, M.: The travelling tournament problem: description and benchmarks. *Lecture Notes in Computer Science*, 2239 (2001), Springer, 580–585
2. Miyashiro, R., Iwasaki, H., Matsui, T.: Characterizing feasible pattern sets with a minimum number of breaks. *Lecture Notes in Computer Science*, 2740 (2003), Springer, 78–99
3. Rasmussen, R.V., Trick, M.A.: A Benders approach for the constrained minimum break problem. *European Journal on Operational Research* (to appear)
4. Trick, M.: Challenge traveling tournament problem. Web Page.
<http://mat.gsia.cmu.edu/TOURN/>, 2006
5. Urrutia, S., Ribeiro, C.C.: Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. *Discrete Applied Mathematics* (to appear)

A Study on the Short-Term Prohibition Mechanisms in Tabu Search for Examination Timetabling

Luca Di Gaspero¹, Marco Chiarandini², and Andrea Schaerf¹

1. Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica, Università di Udine,
via delle Scienze 208, I-33100, Udine, Italy,
2. Dept. of Mathematics & Computer Science, University of Southern Denmark,
Campusvej 55, DK-5230 Odense M – Denmark.
email: l.digaspero@uniud.it marco@imada.sdu.dk schaerf@uniud.it

1 Introduction

Tabu Search (TS) is a well known local search method [11] which has been widely used for solving timetabling problems. Different versions of TS have been proposed in the literature, and many features of TS have been considered and tested experimentally. They range from long-term tabu, to dynamic cost functions, to strategic oscillation, to elite candidate lists, to complex aspiration criteria, just to mention some (see [10] for an overview).

The feature that is included in virtually all TS variants is the so called (*short-term*) *tabu list*. The tabu list is indeed recognized as the basic ingredient for the effectiveness of a TS-based solution, and its behaviour is a crucial issue of TS.

Unfortunately, despite the fact that the importance of a correct empirical analysis has been recognised in the general context of heuristic methods [1,12] and even in the specific case of TS [15], the definition of the parameters associated with the tabu list remains in most research work still a handcrafted activity.

Often, the experimental work behind the parameter setting remains hidden or is condensed in a few lines of text reporting only the final best configuration. Even the recently introduced *racing* methodology for the tuning of algorithms [3] only allows to determine the best possible configuration. This procedures are certainly justified from a practical point of view, but a description of the behavior of the algorithm with respect to its different factors and parameters is surely of great interest in the research field.

In this work, we aim at determining which factors of basic TS are important and responsible for the good behaviour of the algorithm. Instead of the one-factor-at-a-time approach used in [15], our approach uses experimental design techniques [14] combining the racing methodology for the definition of quantitative factors and the analysis of variance for the study of qualitative factors. We focus our analysis on the EXAMINATION TIMETABLING problem, for which there is a consistent literature and many benchmark instances. In particular, we consider the formulation proposed by Burke *et al* [5], which considers first

and second order conflicts (exams in adjacent periods of the same day), but no capacity of rooms.

We plan to extend the analysis to other formulations, and to other timetabling problems so as to have a more general picture of the outcome.

2 Tabu Search Basic Features

At each iteration, TS explores the full neighborhood and selects as the new current state the neighbor that gives the best value of the cost function, independently of whether its cost is less or greater than the current one. This selection allows the search to *escape* from local minima, but creates the risk of cycling among a set of states. In order to prevent cycling, TS uses a prohibition mechanism based on the tabu list. This list stores the most recently accepted moves, so that the *inverses* of the moves in the tabu list are forbidden (i.e., the moves that are leading again towards the just visited local minimum). The two main features related to the tabu list are the following:

Prohibition power: The prohibition power determines which moves are prohibited by the fact that a move is in the tabu list. A move is normally composed of several attributes; depending on the power, the prohibition can be applied only to the move with the same values for all attributes or to the set of moves that have one or more attribute equal to it.

List dynamics: The list dynamics determines for how many iterations a move remains in the tabu list. This can be either a fixed value, or a value selected randomly in an interval, or value selected adaptively on the basis of the current state of the search.

For the EXAMINATION TIMETABLING problem, we consider the search space and the neighborhood relation as defined in [8]. That is, we create one variable per exam with domain equal to the set of periods, and change the value of one single exam at a time. In this setting, a local search move m has three attributes: an exam e , its old period o and its new period n . We identify m with the triple $\langle e, o, n \rangle$.

For the prohibition power, assuming that the move $\langle e, o, n \rangle$ is in the tabu list, we consider the following three alternatives (where the underscore means “any value”):

Strong: All moves of the form $\langle e, -, - \rangle$ are prohibited.

Medium: All moves of the form $\langle e, -, o \rangle$ are prohibited

Weak: Only the single move $\langle e, n, o \rangle$ is prohibited

Intuitively, in the first case, it is not possible to move the exam anywhere in the tabu iterations. In the second case, it is not possible to move the exam back to the old period. In the third case it is not possible only to make the reverse of the tabu move.

For the list dynamics we also consider three possibilities:

Fixed: The tabu list is a queue of fixed size t . At each iteration, the accepted move gets in and the oldest one gets out. All moves remain in the list for exactly t iterations.

Dynamic: The size of the list can vary within the range $[t_m, t_M]$. Each accepted move remains in the list for a number t of iterations equal to $\text{Random}(t_m, t_M)$, where $\text{Random}(a, b)$ is the uniform integer-valued random distribution in $[a, b]$.

Adaptive: The value t depends on the current state. We use the formula (proposed in [9] for graph coloring) $t = \lfloor \text{Random}(0, t_b) + \alpha * c \rfloor$ where t_b and α (real value) are parameters, and c is the number of conflicts in the current state.

3 Experimental Methodology and Results

There are two types of factors present in the analysis: quantitative and qualitative. The two qualitative factors are the prohibition power and the list dynamics, and consist each of three levels. The quantitative factors are the numerical parameters of the list dynamics strategy and may assume an unlimited number of values. There are two issues that complicate the factorial design: (i) the qualitative parameters do not cross with all other factors (for example, there is no α parameter with fixed list dynamics); (ii) the importance of each of the two qualitative factors strongly depends on the values assigned to the underlying quantitative parameters.

In order to understand the relative influence of the qualitative tabu list features a possible way is to split the analysis in two phases. This approach allows to maximise the fairness in the analysis, although perhaps it is not the most economical in terms of number of experiments.

In the first phase, for each combination of qualitative factors, we tune the numerical parameters: $\langle t \rangle$, $\langle t_m, t_M \rangle$, or $\langle t_b, \alpha \rangle$. We tested the following values for the parameters: $t \in \{5, 10, 15, 20, 25\}$, $t_m \in \{5, 10, 15, 20, 25\}$, $t_M \in \{t | t = t_m + 5 \vee t = t_m + 10\}$, $t_b \in \{5, 10, 20, 30\}$, and $\alpha \in \{0.3, 0.5, 0.8\}$.

We perform this task by means of the RACE algorithm developed by Birattari [4]. All experiments are run on 7 instances employed in [5] and each configuration was granted 120 seconds of CPU time on an AMD Athlon 1.5GHz computer running Linux. A t-test ($p < 0.05$) is used to discriminate between the configurations. The best parameter settings found across the tested instances are reported in Table 1. Further details on the racing process will be provided in the forthcoming full paper.

In the second phase, after the values of the parameters have been determined, we perform another experiment whose aim is to understand whether there are main effects or interactions between the two factors, list dynamics and prohibition power, and how these affect the performances of the algorithm. To this aim, we run a full factorial design with 25 replicates per instance and perform an analysis of variance [14]. Each experimental unit exploits a different combination of list dynamics, prohibition power and test instance. In the analysis the test instances are then treated as blocks and hence their influence on the performance

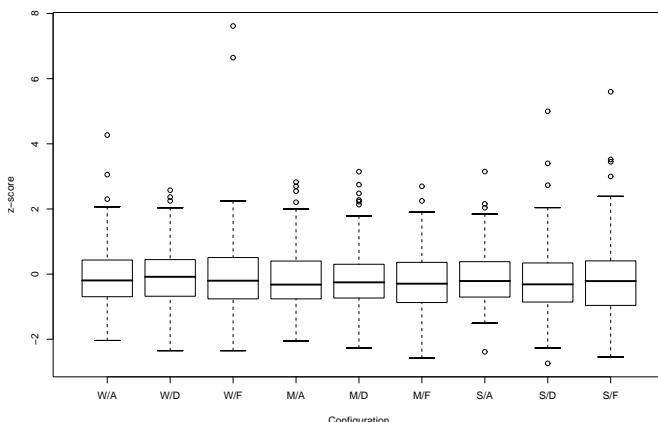
	<i>Weak</i>	<i>Medium</i>	<i>Strong</i>
<i>Fixed</i>	$t = 15$	$t = 10$	$t = 5$
<i>Dynamic</i>	$t_m = 20, t_M = 25$	$t_m = 5, t_M = 10$	$t_m = 5, t_M = 10$
<i>Adaptive</i>	$\alpha = 0.3, t_b = 30$	$\alpha = 0.3, t_b = 30$	$\alpha = 0.5, t_b = 10$

Table 1. Best parameter settings for the various combinations of features

of the algorithms, though recognised, is not taken into account (this entails that the results of the analysis are robust with respect to the set of instances).

Specifically, for the analysis of the $3 \times 3 \times 25$ (<{Strong, Medium, Weak} \times {Fixed, Dynamic, Adaptive} \times replicates) combinations we used two statistical tests. The well known parametric ANOVA, through the F ratio, and the non-parametric Friedman two ways analysis of variance by ranks. Though based on less assumptions, the Friedman test is not able to recognise interaction between the two factors [7]. On the other hand, the F ratio can detect also interactions and is apparently robust even in cases of deviation from the assumptions. In the parametric case we transformed each numerical result, expressed in terms of cost violation of the solution, by standardization of the value within the results per instance. In the non-parametric case the results are instead ranked within the instances, as usual in the Friedman test procedure.

Surprisingly both tests indicated the absence of a significant influence of both main and interaction effects (technically, the F ratio gave a p -value of 0.98 and the Friedman test gave p -value = 0.54).

**Figure 1.** Results of the 9 configurations for the qualitative features (W = Weak, M = Medium, S = Strong, F = Fixed, D = Dynamic, A = Adaptive)

The results are shown in Figure 1 by means of box-and-whiskers plots. A closer insight in the numerical results revealed that indeed there is no important difference in the results. The conclusion is that, if the quantitative parameters are tuned by means of a statistically sound procedure, all configurations of the qualitative parameters perform equally well.

In the future, we plan to investigate the difference in the robustness of the qualitative features, by analysing the sensitivity of the tuning of the quantitative parameters for different configurations. A response surface approach as suggested in [2] would be more appropriate for the selection of quantitative parameters (although it can be computationally more expensive).

We conclude by comparing, in Table 2, our overall best results (in bold) with the currently published ones. From the table we first see that we improved significantly w.r.t. our previous best results ([8]). In addition, although our results are still far from the best ones of Merlot *et al* [13], they are in most cases the second best ones. This improvement is achieved mainly thanks to our statistically sound parameter tuning.

Instance	<i>p</i>	W/A	W/D	W/F	M/A	M/D	M/F	S/A	S/D	S/F	DS [8]	BNW [5]	CDI [6]	MBHS [13]
car-f-92	40	271	292	265	263	278	275	297	224	242	424	331	268	158
car-s-91	51	46	38	38	53	54	32	37	40	33	88	81	74	31
kfu-s-93	20	1148	1027	1103	1047	914	984	1172	1104	932	512	974	912	247
nott	23	112	89	109	103	106	69	109	112	73	123	269	—	7
nott	26	17	17	17	7	15	15	16	13	20	11	53	—	—
tre-s-92	35	0	0	0	0	0	0	0	0	0	4	3	2	0
uta-s-92	38	534	544	526	584	572	507	565	537	567	554	772	680	334

Table 2. Best results found for each configuration and comparison with the best known results.

References

1. R. Barr, B. Golden, J. Kelly, M. Resende, and W. Stewart. Designing and reporting on computational experiments with heuristic methods. *J. of Heur.*, 1:9–32, 1995.
2. T. Bartz-Beielstein. Experimental analysis of evolution strategies – overview and comprehensive introduction. Technical Report Reihe CI 157/03, SFB 531, Universität Dortmund, Germany, 2003.
3. M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Belgium, 2004.
4. M. Birattari. *The RACE pacakge*, April 2005.
5. E. Burke, J. Newall, and R. Weare. A memetic algorithm for university exam timetabling. In *Proc. of ICPTAT-95*, pages 241–250, 1995.
6. M. Caramia, P. Dell’Olmo, and G. F. Italiano. New algorithms for examination timetabling. In S. Nher and D. Wagner, editors, *Proc. of WAE 2000*, volume 1982 of *LNCS*, pages 230–241. Springer-Verlag, 2000.
7. W.J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, USA, third edition, 1999.

8. L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In E. Burke and W. Erben, editors, *Proc. of PATAT-2000*, volume 2079 of *LNCS*, pages 104–117. Springer-Verlag, 2001.
9. R. Dorne and J. Hao. A new genetic local search algorithm for graph coloring. In A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proc. of PPSN V, 5th Int. Conference*, volume 1498 of *LNCS*, pages 745–754. Springer-Verlag, 1998.
10. Fred Glover and Manuel Laguna. *Tabu search*. Kluwer Ac. Publ., 1997.
11. H. H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, 2005.
12. D. S. Johnson. A theoretician’s guide to the experimental analysis of algorithms. In M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: 5th and 6th DIMACS Impl. Challenges*, pages 215–250. AMS, 2002.
13. L. Merlot, N. Boland, B. Hughes, and P. Stuckey. A hybrid algorithm for the examination timetabling problem. In E. Burke and P. De Causmaecker, editors, *Proc. of PATAT-2002*, volume 2740 of *LNCS*, pages 207–231. Springer-Verlag, 2003.
14. D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, sixth edition, 2005.
15. J. Xu, S. Chiu, and F. Glover. Fine-tuning a tabu search algorithm with statistical tests. *Intl. Tran. in OR*, 5(3):233–244, 1998.

A Decomposition Approach with Inserted Idle Time Scheduling Subproblems in Group Scheduling

Cumhur A. Gelogullari¹ and Rasaratnam Logendran²

¹ Operations Research and Decision Support, American Airlines,
4333 Amon Carter Blvd MD 5358 Fort Worth, TX 76155, U.S.A.
cumhur.gelogullari@aa.com

² Department of Industrial and Manufacturing Engineering, 118 Covell Hall,
Oregon State University, Corvallis, OR 97331, U.S.A.
logendrr@engr.orst.edu

Abstract. This paper focuses on minimizing the makespan of a multi-machine flowshop group-scheduling problem that is typically found in the assembly of printed circuit boards, which is characterized as one with carryover sequence-dependent setup times. The intent is to minimize the makespan of schedules comprised of the sequence of board groups as well as the sequence of board types within each group. Specifically, the models and algorithms developed for identifying strong lower bounds on the optimal/near optimal solutions within a reasonable computation time are emphasized. The efficacy of the lower bound developed is demonstrated by using it to quantify the quality of a heuristic solution for the same problem, developed based on tabu search. To obtain strong lower bounds, the problem is decomposed into a master problem and single-machine subproblems which, except for the subproblem on the first machine, are inserted idle time scheduling problems. A tabu search based heuristic is developed to solve the subproblems approximately. Each solution found during the tabu search process is evaluated using a timetabling problem that is formulated as a simple integer program for identifying the inserted optimal idle times on the machine in order to minimize the subproblem objective function. The column generation algorithm developed for the decomposed problem is demonstrated on a real problem obtained from the industry.

Keywords: Carryover sequence dependent setup times, printed circuit board assembly, mathematical programming, tabu search, column generation

1 Introduction, Motivation, and Problem Summary

This paper addresses the multi-machine flowshop group-scheduling problem with carryover sequence-dependent setup times for minimizing the makespan. The problem is typically found in the assembly of printed circuit boards (PCBs) in electronics manufacturing. In order to solve the problem efficiently and effectively, high-level metasearch heuristics based on tabu search have been developed [3]. The

primary focus of this paper is to identify strong lower bounds on the optimal makespan within a reasonable computation time, so that the quality of a heuristic solution, i.e. an upper bound, can be quantified as its percentage deviation from the lower bound. A mathematical programming decomposition approach is developed to obtain strong lower bounds, which, interestingly, involves *timetabling problems* as part of an efficient approximation algorithm used to solve the subproblems in a column generation algorithm.

A PCB is a laminated board assembled with a dozen to thousands of electronic components. The PCB assembly is performed on *automated placement machines* that insert the components on the boards quickly and reliably. Before starting production of a PCB on a machine, the required components are loaded on the appropriate feeders during a setup operation. It is not practical to keep changing the component feeders on a frequent basis for each individual board. Typically, in electronics manufacturing, different board types requiring similar components are grouped together and a single setup operation is performed for each board group. The intent is to load all of the components required of the board types in a group on the proper feeders in *one setup*, just so that the board types in that group can be produced one after the other. As a result, the scheduling problem considered here falls under the category known as *group scheduling* and the scheduling decisions must be addressed at two levels. The problem at the first level is associated with the individual board types within groups and is referred to as the “board level” problem. For the board level problem, a sequence of boards in each group must be determined to minimize the makespan, while different board groups themselves must be sequenced at the “group level” so as to minimize the same performance measure.

The challenges encountered in group scheduling in PCB manufacturing are far greater and distinctly different from that in traditional hardware manufacturing. Although the placement machines automate the PCB assembly processes, high-speed and precise operation and especially the flexibility in tooling makes it a difficult task to control the operations on them. The setup time required of a group of PCBs on a machine is dependent on the configuration of the components on the machines, which in turn depends on *not just on the immediately preceding group*, but on *all of the preceding groups* and the *order* in which they were processed. In a sense, the setup times are not only sequence-dependent, but also *carried over* from the very first board group to the one currently being considered for production. This makes the relationships among setup times of board groups highly complicated compared to the sequence-independent or sequence-dependent setup times encountered in traditional hardware manufacturing. In other words, they can be defined easily in the latter, while the carryover sequence-dependent setup times are hard to explicitly define. In addition, the PCB assembly processes considered here are performed on multiple sequential machines.

2 A Mathematical Programming Decomposition Approach and Timetabling

This paper presents a novel mixed-integer linear programming (LP) formulation of the problem. This problem was addressed by a few researchers previously. However, all of the previous research simplifies the problem either by approximating the setup times as sequence-independent or sequence-dependent setup times, which results in loosing valuable information and identifying inferior solutions. Our model, on the other hand, considers the setup times explicitly, hence introduces a greater degree of accuracy into the problem formulation than before. To obtain strong lower bounds, the problem is decomposed into a master problem and single-machine subproblems which, except for the subproblem on the first machine, are *inserted idle time* scheduling problems [4]. Typically, in an inserted idle time scheduling problem, the objective function is not a regular function and the machine can be kept idle for some time when it could begin processing an operation.

We essentially reformulate the problem as an integer programming problem with an exponential number of variables, each representing a *schedule*. A column generation (CG) algorithm is developed to solve the LP relaxation of the master problem. When solving the subproblems in the CG algorithm, a two-phase approach is employed. In the first phase, the subproblems are solved approximately with a fast tabu search algorithm - tabu search column generator (TSCG) - as long as TSCG identifies new columns. Starting with an initial sequence of board groups and board types within each group, TSCG keeps generating new sequences by performing simple exchange moves. Each sequence (or move) must be evaluated. However, since the subproblems are inserted idle time scheduling problems, it is not straight forward how to evaluate the objective function value for each sequence identified during the search. In this context, therefore, each time a sequence is to be evaluated, we need to *timetable* the board types to insert *optimal* idle times on the machine in order to minimize the subproblem objective function. Such a timetabling problem is formulated as a simple integer program.

It is well known that CG algorithms are prone to the so-called tailing-off effect [1]. While usually an optimal solution is approached considerably fast, it may take a very long time to prove LP optimality. Recently, several methods have been suggested against this drawback. Our approach to stabilize and accelerate the CG algorithm is similar to the one proposed by du Merle et al. [2]. First, a bound is imposed on the dual variables by introducing artificial variables into the LP master problem (LMP) and new negative reduced cost columns are generated until the subproblems fail to do so. If some of the artificial variable values are nonzero, the bound imposed on the dual variables is slightly relaxed at that instance, and the process similar to the one before is continued until all the artificial variable values are zero and no negative reduced column can be identified. In our case, the benefit of introducing artificial variables is two fold: First, the dual variables are initially constrained in some interval so that they smoothly converge to their optimal values. Second, we constrain the dual variables in such a way that the subproblem objective functions - except for the subproblem corresponding to the last machine - become regular and board types can be timetabled without any idle time on the machines. Furthermore, it is shown that

when dual variables are bounded in the subproblems, board types within each group follow the shortest run time rule, which makes the subproblems much easier to solve.

The optimal solution to LMP usually provides a strong lower bound on the optimal makespan, but it is not necessarily integral. Better lower bounds and integral solutions can be identified by branching, resulting in a branch-and-price (B&P) algorithm. The paper develops efficient branching rules that are compatible with the CG algorithm. The idea is based on avoiding generation of sequences in which certain board types are assigned to certain positions on one branch and also allowing them on the other branch.

3 Conclusion

The CG algorithm is demonstrated on a real problem obtained from the industry. The lower bound on the makespan identified by the CG algorithm after only a few iterations is within 2% of an upper bound, and the lower bound obtained from the LP relaxation of the original formulation is shown to be highly inferior to the CG lower bound.

References

1. Barnhart, C., Johnson E., Nemhauser, G., Savelsbergh, M., Vance, P.: Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Oper. Res.* 46 (1998) 316-329
2. du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P.: Stabilized Column Generation. *Discrete Math.* 194 (1999) 229-237
3. Gelogullari, C.A.: Group Scheduling Problems in Electronics Manufacturing. Doctoral Dissertation, Oregon State University, Corvallis, OR, USA (2005).
4. Kanet, J.J., Sridharan, V.: Scheduling with Inserted Idle Time: Problem Taxonomy and Literature Review. *Oper. Res.* 48(2000) 99-110

Multi-Site Timetabling

Ruben Gonzalez-Rubio

Département de génie électrique et de génie informatique
Université de Sherbrooke,
Sherbrooke, Québec, J1K 2R1
Canada
Ruben.Gonzalez-Rubio@USherbrooke.ca

Keywords : Timetable Construction, Timetable algorithms, Timetable software, DIAMANT.

1 Introduction

Timetable construction in Universities is a moving target. De Werra [1] indicates that educational methods are driving changes in models and software applications, helping or constructing timetables. In this article we present a new timetable model which came from a new program in one Faculty¹ at the Université de Sherbrooke.

The basic difference of this new program compared with other instances of timetabling constructions in the University is that the courses in a term are taught at different sites instead of only one. In the case of one site all rooms are located in a single building, whereas in this new department the courses take place in three different sites. These sites are located in three cities (the longest distance between two sites being about 150 km). Some professors can teach at one site, some others can teach at two sites. For the moment nobody teaches at the three sites, but it could change, which means that the algorithm and the program must handle this possibility.

One last special constraint is that in two sites there is one teleconference room, which means that in some cases a professor can give a lecture in both sites at the same time. This implies that the lecture can be taught to two sections at the same time.

The article is organized as follows: First we introduce the current model for courses timetabling, then we show the new model to make clear where the differences are. Next, we present the steps that we follow to integrate the functionality of multi-site problem. After that we present how DIAMANT was modified to take into account the new data and the new algorithm, followed with a discussion of the experience. We end with a conclusion.

¹ A program in this context means an entity that can deliver a degree, the entity is a department, in this case a program can deliver degrees in various specialities.

2 The Multi-site model

2.1 The Université de Sherbrooke timetabling model

Currently DIAMANT is used to schedule timetables at Université de Sherbrooke. DIAMANT [2] is an iterative application developed at the University which helps in scheduling courses and exams timetables. In this article we only talk about how to schedule courses.

Various definitions of the timetabling problem exist [5], [6] and [4]. We prefer that of fixing in time and space a sequence of meetings between teachers and students, in a prefixed period of time.

At Université de Sherbrooke there are many entities preparing timetables, each entity producing its own timetable using DIAMANT. Some entities work as demand driven systems, the students select courses from a list, then the schedule is prepared. Other entities prepare a master timetable, then the students can select the courses they want to take.

A more detailed definition of the problem could be the scheduling of a set of lectures for each course with a given number of rooms and time periods. For each set of lectures there is a professor. A professor can teach more than one course. Each entity has a prefixed period of time.

We formulate the problem following the presentation in [1], the we introduce changes to handle unavailabilities and preassignments. We have also the multiple sections and grouping subproblem and the classroom assignment subproblem. The periods are of variable length. The period sizes are different from one entity to an other.

2.2 The Université de Sherbrooke multi-site timetabling model

The multi-site problem came from an entity at the University, this entity offering courses (lectures) in three (sites) cities, the distances between cities being about 150 km, 90 km and 60 km. A set of courses is assigned to each site. That means a lecture of a course will take place on the specified site. A course can have multiple sections, sections can have lectures in the same site or in different sites. The students are assigned to each site, a set of rooms is defined for each site, a subset of professors can teach at one site, another subset can teach at two sites. For the moment nobody teaches at the three sites, but it could happen in a future.

The professors are assigned to a course or a section. The schedule must respect all hard constraints (no two classes in the same room, and so on). A new special constraint came from the fact that the travel time from one city to the other makes it impossible to teach two classes in a row at two different sites.

3 Steps to solve the multi-site problem

In order to solve de problem we proceed as follows:

1. Simplified solution with no changes to the application, we recall that the application can work manually or automatically.
 - (a) We divide the problem into three problems (three sites). This allow to use the application three times to build three independent schedules.
 - (b) As the three problems are dependent, a professor assigned to a period becomes unavailable elsewhere and the next, to let the professor travel from one site to the other. Changing the professor availability is done manually, when the schedule of the other site is prepared.
2. Solution with changes into the application.
 - (a) One problem instead of three. The resources (courses, rooms, and students) are assigned to the corresponding site. As a professor is assigned to a course, he is indirectly assigned to a site. Inside the application we can have three sub problems. A menu allows the user to consult the state of a schedule of each site.
 - (b) Professor availability can be calculated for each site by taking into account if he has a lecture in another. The unavailabilities are displayed in different colors according to their site.
 - (c) The user can fix the sequence of building the schedules of each site. The sequence is executed automatically.
 - (d) Teleconference room problem is solved by the new class Teleconference-Room which is a derived from class Room.

4 Changes in the application DIAMANT

DIAMANT was build using object-oriented technology and design patterns [3] in order to simplify modifications. There are classes representing a set of resources (courses, professors, rooms, and students). We modify each class representing a resource to take into account the new data, for example we add a data member to Student Class to indicate the site where the student follows a course.

While reading the data the application detects that a multi-site problem must be solved. Then the menus are updated to work in multi-site mode.

Some dialogs were also modified to take into account the new data members.

We implemented the pattern Strategy to choose dynamically the algorithm to be executed. When there is more than one site the multi-site algorithm is executed. The multi-site algorithm follows the sequence of schedules fixed by the user.

5 Experience

When we saw the requirements for the multi-site problem, we thought that it would be difficult to change the application. Using an iterative development process and taking advantage of our object-oriented design, we succeeded in a short period of time to satisfy our users. The whole development took about four months.

6 Conclusions

We introduce the problem of multi-site timetabling. We presented how we modified the application in order to treat a new the problem. Our the decision of developing using object-oriented programming really pay off. This allows us to add new functionalities without disturbing users that do not use these new functionalities. The new multi-site functionality was added in a short period of time.

References

1. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.
2. R. Gonzalez Rubio. Generating university timetables in a interactive system: DIAMANT. In *PATAT'00*, Konztanz, Germany, August 2000.
3. R. Gonzalez Rubio and Y. Syam. A design pattern: “TestConditions” which could be used in timetable construction software. In *PATAT'02*, Gent, Belgium, August 2002.
4. Luís Paulo Reis and Eugenio Oliveira. A language for specifying complete timetabling problems. In Edmund K. Burke and Wilhelm Erben, editors, *PATAT*, volume 2079 of *Lecture Notes in Computer Science*, pages 322–341. Springer, 2000.
5. A. Schaerf. A survey of autamated timetabling. *Arificial Intelligence Review*, 13(2):87–127, 1999.
6. Anthony Wren. Scheduling, timetabling and rostering - a special relationship? In Edmund K. Burke and Peter Ross, editors, *PATAT*, volume 1153 of *Lecture Notes in Computer Science*, pages 46–75. Springer, 1995.

Scheduling the Belgian Soccer League

Dries Goossens and Frits C.R. Spieksma

Department of Operations Research, Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven, Belgium
dries.goossens@econ.kuleuven.be

1 Introduction

Especially in Europe, soccer has become big business, involving many parties (e.g. teams, police, broadcasting companies, ...) and a lot of money. Naturally, the schedule of the matches is of great importance, since it has a considerable impact on the costs or revenues of all parties involved. Each party has its (possibly conflicting) constraints and wishes, which makes it hard to generate a schedule that is considered fair and acceptable to all parties. In literature, there are papers on the scheduling of the national soccer league of various countries, e.g., Germany [1], Italy [2], The Netherlands [5] and Austria [1]. However, because of some specific constraints that characterize each of these competitions, the models presented in these papers are not readily applicable to soccer league scheduling problems in other countries. In the next section, we give an overview of the parties involved in scheduling the Belgian Soccer League and their requirements.

2 Problem description

The Belgian Soccer League consists of 18 teams, which play a double round robin tournament (i.e., each team plays against each other team twice). The league is intermural, meaning that a team plays in its own stadium or in the stadium of the opponent. Furthermore, for reasons of fairness, the schedule in the second half of the competition should be the same as in the first half with opposite home away pattern (i.e., mirroring). The 34 weekends on which there is a matchday are given by the Royal Belgian Football League (KBVB).

Several other requirements must be taken into account. These requirements are based on conversations we had with a representative of the KBVB. First of all, two teams play in the same stadium and therefore, for each week, if one of these teams plays at home, the other should play an away match and vice versa. The same goes for another pair of teams that share the same advertising panels. Also, a mayor can forbid that a game is played in his or her town on one or more dates. The reason behind this is usually that there is some other event (for instance, a summit with EU leaders) in his town needing the attention of the local police. Furthermore, there are a number of so called "risk matches" for which national police is needed to maintain order. For reasons of security, risk

games can also be forbidden for given dates and locations. Another requirement is that teams should not play more than two consecutive home (away) matches. Finally, the rights to broadcast live matches are sold to a company that in return requires a calendar with at most one "top match" (i.e., games between two of the four "big" clubs in Belgian) per matchday and at least one top match in the final three matchdays (in order to have a competition that is thrilling until the end).

Apart from these hard constraints, there are quite a number of soft constraints. Matches are normally played on a Saturday, however, the broadcasting company has the right to shift a match to Friday and a match to Sunday on a month's notice. Since this could be problematic for teams that also play a mid-week match (e.g. Champions League), matches between those teams should be scheduled as much as possible on weekends that are not preceded or followed by a midweek match. Furthermore, each team has its wishes regarding the calendar. For instance, some teams prefer not to play at home when some other team plays at home, because they are afraid that a part of their spectators would attend the other game. Also, most teams prefer not to play against all top teams consecutively, nor do they like to welcome all top teams at home during the first half of the season. Finally, the number of breaks (i.e. consecutive home (away) matches for a team) should be minimized. The scheduling problem is to decide for each matchday which teams plays against each other and which one of each pair plays at home, satisfying all of the hard constraints and as many soft constraints as possible.

3 Solution approach

Currently, this problem is solved by a scheduler of the KBVB who starts from a so-called basic match schedule (BMS). A BMS gives for each team a home-away assignment and the opponent. For instance, if the BMS for team A looks like $-B + E - F + J\dots$, this means that team A plays away against team B on the first matchday, at home against E on the second, and so on. The BMS that is used is a solution to the double round robin problem with perfect mirroring and minimizes the number of breaks. The teams are then each assigned to a letter in the BMS, taking into account the constraints. The scheduler solves this assignment problem by hand. With a lot of patience and a 30-year experience, this results in a schedule that satisfies most of the hard constraints, but disregards the soft constraints. Not surprisingly, during the last years, a number of teams have expressed their displeasure with the league schedule, with some of them even calling it unbalanced (see [3]).

Obviously, there is room for improvement in assigning teams to the letters from the BMS. We developed a mathematical formulation for this problem, which tries to satisfy the hard constraints (in so far that they are not conflicting). Each of the soft constraints were given a penalty by the KBVB, so that the formula-

tion minimizes the total penalty of the violated constraints. The model can be solved using Ilog Cplex within the hour and was used to construct the schedule for the season 2006-2007.

Although widening the search to multiple basic match schedules will typically lead to a better calendar, the KBVB has decided to stick with the current BMS in order to get a schedule that is compatible with those of the lower divisions. However, it is our intention to investigate the possibilities of considering other basic match schedules with regard to next year's calendar.

References

1. Bartsch, T., Drexl, A., Kröger, S.: Scheduling the professional soccer leagues of Austria and Germany. *Computers & Operations Research* **33** (2006) 1907–1937
2. Della Croce, F., Oliveri, D.: Scheduling the Italian Football League: an ILP-based approach. *Computers & Operations Research* **33** (2006) 1963–1974
3. G.L.R.: Voetbalkalender eerste klasse 2005-2006: Genk boos. *De Standaard* (2005) June 15th
4. Nemhauser, G. L., Trick, M.A.: Scheduling a major college basketball conference. *Operations Research* **46**(1998) 1–8
5. Schreuder, J.A.M.: Combinatorial aspects of construction of competition Dutch professional football leagues. *Discrete Applied Mathematics* **35** (1992) 301–312

A Four-phase Approach to a Timetabling Problem in Secondary Schools

Peter de Haan, Ronald Landman, Gerhard Post, and Henri Ruizenaar

Department of Applied Mathematics, University Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands

1 Introduction

Timetabling problems are present in all types of schools. The research in this area is still very active; of the 19 selected contributions of PATAT 2004 ([1]), 12 are dedicated to Educational Timetabling. These problems can often be modeled by a graph coloring problem. Here the vertices represent the events (lessons, courses, exams) to be scheduled, the (vertex)colors the available timeslots, and the edges express incompatibilities between two events. Typically incompatibilities are caused by the effect that resources related to an event (teachers, students, rooms) can attend at most one event at the same time.

Apart from the basic model of graph coloring, extra conditions are common. Typical conditions are (room) capacities, resource (room, teacher) availabilities, and precedence relations among events.

The subject of our study is a High School Timetabling Problem as it is common in the Netherlands. Beforehand it is decided which teachers give which lessons. Hence the events are the lessons, and, in principle, the problem can be stated as a graph coloring problem with extra conditions on availability of resources (rooms, teachers). However there is an extra dimension: the quality of the constructed timetable. A feasible solution, though necessary, is absolutely not sufficient: we need to improve the feasible timetable to a schedule that is acceptable to use.

The size of the graph involved, and the extra efforts to improve the quality are the main reasons for our 4-phase approach: we try to control the quality by a preprocessing phase, and a post-processing phase. In the preprocessing phase, we cluster events in so-called clusterschemes. These clustered events can be considered as the new events to be scheduled. In the second and third phase a feasible timetable is constructed. In the fourth phase a Tabu Search is used to improve the best schedule found. The developed approach is tested by using data from the Kottenpark, which is one of the locations of ‘Het Stedelijk Lyceum’ in Enschede, the Netherlands.

2 Problem description

In the Kottenpark the timetable is still made by hand, and checked by computer. The reason for not using the automatic planner, that their commercial package provides, is mainly quality: this package is not able to generate any complete solution, and moreover the part that is generated is of bad quality.

In 2004, the Kottenpark had around 1000 students, 36 school classes, 71 teachers, and 40 rooms. There are 1049 lessons to be scheduled. As such it is a school of average size in the Netherlands. A bottleneck is the number of rooms for physical exercise, which is only 2, accounting a 100% occupation. All lessons have to be scheduled in 37 timeslots: 8 timeslots on Monday, Tuesday and Wednesday, 6 on Thursday (2 hours are reserved for staff meetings), and 7 on Friday. The classes have a timeslot occupation ranging from 73% up to 97%.

There is one major complication in the Dutch situation; for students in the upper years, 2/3 of their lessons are in optional subjects. This means that the basic class-teacher model (see for example [4]) can not be used anymore. We cope with this situation by using clusterschemes, see Phase 1 in the next section.

The most important aspects, which were included in our research, are the following:

- The lessons of a subject should be on different days.
- Some lessons should be given as a block. A block means that the subject is scheduled on two consecutive timeslots on the same day. This is often the case for physical exercise, but can also be the case for other subjects.
- Some teachers are not available on specific days, or parts of days.
- The lessons of some teachers should be concentrated on a limited number of days.
- In the lower grades the schedule must be compact (without free periods); in the higher grades free periods should be avoided.
- Free periods should be avoided for (most) teachers.

3 The four phase approach

Our approach consists of four principal phases.

Phase 1. Constructing the clusterschemes.

Students in the upper grades have optional subjects. All the groups for the optional subjects of one grade (and level), are put in a clusterscheme. This clusterscheme is divided in clusterlines, and students are to be placed in groups, such that the groups within one line have no students in common. Consequently all the subject groups in one line can be scheduled at the same time. Working with clusterschemes is not new (see for instance [2, 3]), and is in use at (nearly) all secondary schools in the Netherlands. We use a branch & bound algorithm to solve this problem.

Phase 2. Assigning lessons to day-parts.

In second phase the lessons of clusters of subject groups are assigned to day-parts one by one. The method we use is dynamic priority rule. At each stage we estimate the difficulty for a cluster to be scheduled. This estimate is mainly based on the availability of the resources. We schedule the cluster with the lowest availability first. If a resource gets tight on a day-part, we construct a graph with lessons of this resource on this day-part, and check whether these lessons can still be scheduled.

If this scheduling process breaks down (a particular cluster can not be assigned anymore), the heuristic lowers the availability (or increases the priority) of this cluster, and starts all over again. After a few tries, all lessons are assigned to day-parts, and we can proceed to the next phase. Because of the checks in this phase we expect that there exists a feasible solution for Phase 3.

Phase 3. Scheduling the day-parts.

The third phase schedules the lessons on day-parts to timeslots. Hence the original graph with all lessons is broken down to 10 smaller subgraphs. We use a graph coloring heuristic, which colors the points one by one (first fit). Originally we use the degree to sort the points, but we also use random orders to obtain several schedules.

Phase 4. Improving the schedule.

We use a Tabu Search to improve the best schedule found in Phase 3. For this we determine the worst resource (teacher or school class). Here ‘worst’ is measured by the objective function. For this resource we consider all lessons and free timeslots and determine the 10 best (lesson, new timeslot) combinations, as far as the worst resource is concerned. Starting with such a combination we consider an ejection chain: rescheduling a lesson of the worst resource to a new timeslot may be infeasible regarding the *other* resources of this lesson. We try to lift this infeasibility, by shifting the conflicting lesson to another timeslot, which can cause infeasibility, etcetera. We do not use branching of possibilities here, and stop if the chain of shifts reached a given length.

If one or more of the (lesson, new timeslot) combinations can be rescheduled, we perform the best chain of moves. We use tabu lists for lessons and resources to make sure that the next step does not undo the current step.

4 Results and conclusions

The presented study is performed with data from a specific Dutch school, but we believe that this data is representative for many schools in the Netherlands. Unfortunately not all constraints were incorporated yet, which makes comparison to the real timetable not completely fair. Comparing what *was* included we see a huge improvement in quality; for instance the number of free periods for teachers drops from 128 (hand-made) to 48. Using additional interactive methods, the quality can be improved to a level hard to obtain by manual scheduling.

References

1. E. Burke, M. Trick (eds.), *Practice and Theory of Automated Timetabling V*, Lecture Notes in Computer Science 3616, Springer Verlag, Berlin (2005).
2. B. van Kesteren, ‘The clustering problem in Dutch high schools: changing metrics in search space’, Master’s thesis, Leiden University, The Netherlands (1999).
3. J.L. Simons, ‘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).
4. D. de Werra, *An introduction to timetabling*, European Journal of Operational Research **19**, (1985), pp. 151-162.

Framework for negotiation in Distributed Nurse Rostering Problems

Stefaan Haspeslagh¹, Patrick De Causmaecker¹, and Greet Vanden Berghe²

¹ K.U.Leuven Campus Kortrijk, Department of Computer Science,
E. Sabbelaan 53, 8500 Kortrijk, Belgium

{Stefaan.Haspeslagh,Patrick.DeCausmaecker@kuleuven-kortrijk.be}

² KaHo St.-Lieven, Information Technology,
Gebr. Desmetstraat 1, 9000 Gent, Belgium
Greet.VandenBerghe@kahosl.be

Abstract. This contribution deals with the distributed version of the nurse rostering problem. It is considered in a hospital with many separated wards. The nurse rostering problem within a ward is ‘the local problem’, rosters within a ward are ‘local rosters’. At any time in the process, i.e. at the time of the construction of a roster for a certain time period as well as in the course of this time period when unexpected events cause rescheduling, wards may call in the help of their peers. At this level negotiation will take place. The details of the local rosters do not necessarily enter this negotiation level, and when they do, they may be translated from a local representation to a generally accepted vocabulary. After a motivation for a distributed approach, a general architecture is proposed and a negotiation protocol is described.

1 Introduction

The Nurse Rostering Problem (NRP) has been the subject of intensive study. The literature is very extensive and an overview is presented by Burke et al. [1]. The NRP is by nature a highly constrained problem and very hard to solve. Much attention has been paid to the local problem, i.e. at ward level. Rosters typically have to satisfy each nurse’s work regulation while realizing a given coverage. Nurse preferences must be satisfied if possible. Apart from generating rosters for a fixed period of time, sudden staff shortages due to absence or unexpected overload must be coped with.

In the presented approach, we allow for requests to be raised by a ward in case of shortage. Other wards consider re-rostering in order to satisfy this demand. They submit weighted proposals to the requesting ward, the weight representing a cost associated with the rearrangement. When the specific request cannot be resolved, alternative proposals can be formulated. For instance, suppose that ward x signals a shortage of a nurse with a certain skill level within a particular shift and suppose that ward y cannot satisfy this request. Ward y can decide in that particular case to offer a nurse for another shift or with an equivalent skill

level. Similarly, a request may specify a number of alternatives. We discuss this approach in more detail below.

Section 2 presents a brief overview of related research. Real world requirements related with the Nurse Rostering Problem are given in section 3. Using those requirements, section 4 argues why a distributed approach is appropriate. Section 5 investigates the criteria used to evaluate the rosters at each level. In section 6, we propose the ‘General Architecture’ designed to tackle the problem. Section 7 elaborates on an initial implementation. Finally, plans for future research are discussed.

2 Relation to published work

The nurse rostering problem that involves staff shortage, is often solved by re-planning under temporarily relaxed constraints. Several other solutions have been studied. Trivedi and Warner [4], for example, introduce a pool of *float nurses*. Siferd and Benton [5] resolve personnel shortage by calling nurses from other wards and by allowing nurses to work overtime.

At the level of the hospital, the size and the complexity of the problem become much larger. However, moving up to this level is not only a matter of scale. Some concerns disappear and others come up. Each ward, e.g., will typically maintain its own interpretation as to which extent certain constraints - not only preferences but work regulations as well - must be satisfied. Trying to solve the NRP at this level with the methods described in [1] is much harder if feasible at all. In the present contribution, we study the application of negotiation to tackle this problem.

This approach was pioneered by Kaplansky and Meisels in [6]. They introduce three stages. Scheduling agents generate a local solution in the first stage. The second stage constructs a globally feasible solution. The last stage is used by the scheduling agents to improve their local solutions. Kaplansky and Meisels assume a fixed pool of shared resources. Within our framework, all nurses are available for exchange.

Bard and Purnomo [7] introduce several ways to cope with personnel shortage. We mention those that are relevant for the present discussion. Initially, negotiation takes place with a pool of float nurses. Consequently, when a ward has more nurses than needed, the remaining nurses are placed on an on-call list. Other wards can call for those nurses when personnel shortage occurs. Finally, agency nurses are used. Replanning never takes place.

3 Real world requirements

The purpose of this section is to state the real world requirements in a hospital. In the next section, we use these requirements to show that they are naturally met by a distributed system.

A first class of requirements have to do with the scale of a hospital wide system and its associated maintainability problem. The second class of requirements

stem from the kind of information that is available either at the local or the global level and the inherent communication and interaction problems at the different levels.

The scale of a ward allows to build systems that can be used as real-time rostering systems [2]. This allows to react promptly to unexpected unavailabilities or temporary work overload. The manager can use the system for on-line consultancy and immediately discuss the possibilities with his staff. While hospital wide systems can be built [3] the execution times and the flexibility of the interface cannot be expected to allow for this kind of what-if analysis unless provisions are made to do one-ward-only rescheduling. This is closely related to the maintenance problem when moves of personnel cause changes in the local constraints and preferences.

The kind of information being processed at a local ward does not always have relevance or meaning at the hospital level. A specific nurse may not like to work on Thursdays or for interpersonal reasons two nurses should not make a team too often. This gives rise to preferences that a ward manager will want to take into account without spreading it throughout the hospital. A similar requirement goes for the work regulations. Although official individual work regulations are recorded at a central level, it might not make sense to have a central rostering system dealing with those regulations when building a hospital wide work roster. One rather wants these details to remain encapsulated at the ward level while reasoning about hospital wide concerns at a separate level. Among these concerns may be a fair distribution of resources and work load or strategic intention to favor a certain ward in order to increase a quality level or to handle a chronic safety problem.

We can thus conclude that a hospital nurse rostering system has to take at least two levels of decision making into account and that the criteria and objectives at the levels are essentially different. We argue in the next section that this requirement is naturally met by a distributed rostering system.

4 Motivation

In this section we describe the properties and benefits of a distributed method. First of all, a distributed approach offers a good *scalability*. When a new ward is organized, little effort must be made to support the new ward. Second, such a system is *flexible* and in a sense *easy to implement*. It can be introduced step by step. A couple of wards could initially implement the system. The system can then be gradually expanded with the other wards, but not every ward has to join. The way rostering is performed locally may differ at each ward. This way a large *autonomy* is provided to the wards. Our rostering approach also allows a large amount of communication between the wards. Since decisions are communicated, they might be more easily accepted.

5 Evaluation criteria

Following the two level example from section 3 we investigate the evaluation criteria and objectives that will be valid at each of the levels.

At the level of an individual ward, the criteria are those that have been analyzed in previous studies [2]. Broadly those are the satisfaction of the demands and the constraints with (individual) work regulations. Demands stem from the workload per time unit or shift in the ward and are typically expressed as a number of nurses of a specific qualification. Work regulations define, sometimes nurse-specific, values for e.g. the maximum number of consecutive night shifts, the minimum number of consecutive days off, and so on ... Sometimes patterns are used to specify preferred work time distributions. These constraints are often very complicated and their modeling is a highly demanding task. In addition one wants a system that allows to specify personal preferences and ad hoc requests. We revisit implementation issues in the light of the present study during the presentation.

At the hospital level other criteria must be taken into account. At this level the manager wants to guard the fairness of the work load distribution, wants to raise certain quality levels, wants to decide where resource shortages are allowable at peak moments and so on ...

A model must allow to express the requirements at all levels. In this contribution we will discuss a negotiation based model. Wards will be represented by autonomous agents. They will generate rosters along well investigated lines, but they will be able to detect a local shortage and to formulate requests to solve shortages. The agents will also be able to interpret requests and generate answers to requests. Our negotiation protocol is designed in such a way that it allows to express local needs at the higher level, the hospital level. The decision criteria for individual wards to respond to requests and to accept proposals are used to express the hospital level requirements. We develop an example during the presentation.

6 General Architecture

The General Architecture consists of three building blocks. A first building block is a common language. This language enables the wards to formulate internal information into a representation that each ward understands. The second building block is a negotiation protocol. This protocol allows a ward to formulate multiple questions and to select one of the alternatives offered. The third building block represents a ward. Each ward (e.g. W_1 , W_2 and W_3) is an autonomous entity that consists of the following components:

- IDP: When a problem in a ward occurs, the Internal Problem Descriptor specifies the problem.
- EDP: The External Problem Descriptor translates an internal problem into a request in terms of a common language known by all wards.

- ERD: The External Request Descriptor transforms an incoming request into an internal representation.
- IRD: The Internal Request Descriptor passes the request to the internal engines.
- IAD: A proposal to meet the request is formulated into an answer by the Internal Answer Descriptor
- EAD: The External Answer Descriptor translates the answer into a common language.
- EOD: An offer by a ward is translated into internal terms by the External Offer Descriptor
- IOD: The Internal Offer Descriptor handles offers.

They will be discussed in future work.

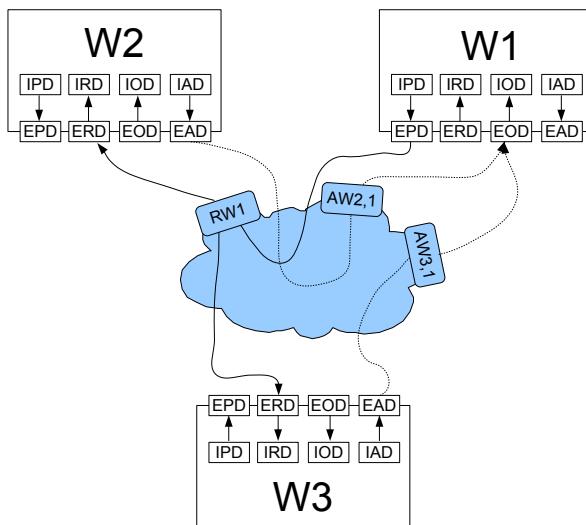


Fig. 1. General Architecture

An overview of the architecture is presented in Fig. 1. RW_x denotes a request by Ward x , and $AW_{y,x}$ denotes Ward y 's answer to Ward x .

7 Initial solution

In a first iteration of this problem, two building blocks are worked out. The common language allows wards to raise specific requests in case of personnel shortage and to submit proposals to satisfy demands. Particular attention is paid to the assurance that minimal information needs to be exchanged for formulating requests and answers. Secondly, a negotiation protocol is designed. In order to

test our approach, a ward is elaborated with simplifications of the P- and R-components.

8 Future research

Our initial common language supports full answers to requests. Further elaboration must allow for partial proposals to satisfy requests. A more sophisticated negotiation protocol needs to be worked out. Similarities and repetition of requests should be analyzed. E.g. if ward x always lacks a nurse for a certain shift, and ward y is always able to respond to this demand, a transfer of personnel from y to x may be considered.

Intelligence must be built into the components introduced in Section 6. For example, an internal request by the *EPD* could be translated in different versions destined for different wards. The *EPD* can *learn* how a particular ward responds to certain requests and attempt to formulate the requests in such a way so that the ward will certainly respond.

References

1. E.K. Burke, P. De Causmaecker, G. Vanden Berghe, H. Van Landeghem. The State of the Art of Nurse Rostering. In *Journal of Scheduling*, 2004, Vol. 7, No. 6, 441-499 (2004)
2. E.K. Burke, P. De Causmaecker, G. Vanden Berghe: Novel Meta-heuristic Approaches to Nurse Rostering Problems in Belgian Hospitals, Chapter 44 in J. Leung: *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, CRC Press, 2004, 44.1-44.18
3. E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe: Metaheuristics for handling Time Interval Coverage Constraints in Nurse Scheduling, *Applied Artificial Intelligence*, Vol. 20, No. 3, 2006 (to appear)
4. Trivedi V.M. and M. Warner. A branch and bound algorithm for optimum allocation of float nurses. In *Management Science*, 22(9), 972-981 (1976)
5. Siferd S.P. and W.C. Benton. Workforce staffing and scheduling: Hospital nursing specific models. In *European Journal of Operational Research*, 60, 233-246 (1992)
6. E. Kaplansky and A. Meisels. Distributed Personnel Scheduling - Negotiation among Scheduling Agents. In *Annals of Operations Research*, (to appear)
7. J.F. Bard and H.W. Purnomo. Real-Time Scheduling for Nurses in Response to Demand Fluctuations and Personnel Shortages. M.A. Trick and E.K. Burke (editors) Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, Pittsburgh, 67-87 (2004)

Scheduling Research Grant Proposal Evaluation Meetings

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 experts to be unavailable for the entire duration of the meeting; constraints of this nature, and others complicate the search for a solution. We report on a system developed to ensure the smooth running of such meetings.

1 Background

The process of evaluating research grant proposals presents some interesting opportunities for operations research practitioners and researchers. The model we discuss here assumes that a fixed pool of evaluators exists and the set of grant proposals is distributed amongst them subject to the evaluators' stated abilities to evaluate each. (Although this step of the process is outside the current scope it is an interesting assignment problem with many side constraints. For example, some evaluators, such as vice-chairs, may be expected to take a reduced load of evaluations due to other duties; no proposal should have a majority of vice-chairs evaluating it; and, 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.)

In the present context our interest begins when a general panel¹ meeting 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 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

¹ A panel may be thought of as a general research area, *e.g.* computer science.

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 expert 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 experts 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 bears some resemblance to the teacher-class timetabling problem, the objective function differs, as well as some of the constraints. In the following section we describe our first approach to solve the problem by modelling it as an ILP.

2 An ILP Model

Given $E = e_{ij}$, the assignment matrix that indicates what experts 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 expert 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 expert 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 expert is assigned to a time slot then s/he must be evaluating exactly one proposal from their allocation and, conversely, if an expert is not assigned to a time slot then none of their allocation are being evaluated in this slot.

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

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

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

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

The goal is to minimise the number of time slots used. To do this we ask “how many proposals were evaluated in time slot j ?” ($\sum_i x_{ij}$) and charge j for

each of these. This forces as many proposals as possible into “small” time slots for, if a proposal is evaluated in a later time slot j' , instead of j , the cost rises by an amount $j' - j$. The cost z is

$$z = \sum_j (j \sum_i y_{ij})$$

Thus the ILP model we solve is

$$\text{minimize} \quad \sum_j (j \sum_i y_{ij}) \quad (1)$$

subject to

$$\sum_j x_{ij} = 1 \quad \forall i \quad (3)$$

$$y_{ij} = \sum_k e_{ki} x_{kj} \quad \forall i, j \quad (4)$$

$$w_i x_{ij} \leq \sum_k e_{ik} y_{kj} \quad \forall i, j \quad (5)$$

x_{ij}, y_{ij} binary

For a problem instance involving 58 evaluators and 383 proposals using CPLEX 7.0 and running on a Pentium V, the previous model had not terminated after 3 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

Graph coloring is long associated with timetabling [2–4] and we adopt this approach here. For the model presented in Section 2 we construct a graph $G = (V, E)$, 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 $S_i, i = 1, \dots, C$.

Using a Tabu search-based vertex colouring algorithm gives quite satisfactory results on problem instances commensurate with that described earlier.

3.1 Restricted Evaluator Availabilities

Restricted availability of one or more evaluators can be accommodated by solving a *maximum cardinality bipartite matching* instance [1]. We construct the bipartite graph $B = (U, V, E)$, where $U = \{i | 1 \leq i \leq C\}$ is the set of colourings and $V = \{j | 1 \leq j \leq T\}$. Vertices u and v are connected by an edge if it is possible to schedule *all* evaluators involved with proposals P_u during time period v . The neighbourhood of u is the set of time slots in which all proposals P_u may be feasibly scheduled.

However, two proposals assigned to the same colour class may require evaluators who cannot be present simultaneously and this will result in vertex u having 0 neighbours, and thus unschedulable. Therefore, it is necessary to add to G , prior to colouring, an edge between every pair of proposals having evaluators not present simultaneously. (A separate but related *feasibility* check ensures that if two evaluators work together on k proposals then there are at least k slots when both are available.

Soft Constraints Constraints R2 and R3 are treated differently since they do not affect feasibility. We build the bipartite graph as previously described 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 a Pentium V 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.

4 Discussion

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 than heretofore.

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 loose opportunities for finding solutions that are more satisfactory with respect to the soft constraints.

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.

References

1. Helmut Alt, Norbert Blum, Kurt Mehlhorn, and Markus Paul. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1.5}\sqrt{m/\log n})$. *Inf. Process. Lett.*, 37(4):237–240, 1991.
2. E.K. Burke, K.S. Jackson, J.H. Kingston, and R.F. Weare. Automated timetabling: The state of the art. *The Computer Journal*, 40(9):565–571, 1997.
3. C. Friden, A. Hertz, and D. de Werra. STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.
4. D. C. Wood. A technique for coloring a graph applicable to large-scale timetabling problems. *Computer Journal*, 12:317 – 322, 1969.

Making good rosters for the security personnel

Han Hoogeveen, Eelko Penninkx

Institute for Information and Computing Sciences,
Universiteit Utrecht, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.
e-mail: slam@cs.uu.nl, penninkx@cs.uu.nl

1 Problem description

We look at the problem of finding good rosters for the security personnel of the academic hospital Utrecht Medisch Centrum (UMC) in Utrecht. The security personnel have to man several posts in the hospital, seven days a week, 24 hours a day. Therefore, the people work in three shifts: Morning, Late, and Night; the required minimum number of persons varies per shift. Except for a number of part-timers, each worker has a contract for 36 hours per week, but they are assigned approximately 34 hours per week on average to compensate beforehand for extra work that has to be carried out to replace people that are ill. Next to the ordinary working shifts, there are the so-called stand-by shifts, during which a worker can be called as a replacement. Finally, the workers have to follow a training session once-in-a-while, which is done groupwise during the Wednesday morning shift. Our task is to produce a set of good rosters for a one-year period. These rosters have to satisfy several regulations, which decree for instance that the number of hours worked per day should be reasonably balanced each week, that the number of consecutive Night shifts is at most 4, and that there should be enough time off after the last Night shift. Furthermore, there is a demand to consider rosters in which the shifts follow the order Morning-Late-Night. Finally, the rosters have to be personalized, such that they reflect the personal preferences as much as possible; we wish to maximize the total satisfaction, with the side-constraint that the unluckiest person is not extremely unlucky. Note that in the current situation there are standard rosters: in week 1, person i gets roster i , and when the week is over, he moves up to the next roster in the list, until the whole cycle of weeks has been run.

2 Solution approach

Here we discuss our initial solution approach; the problem has been changed recently (see Section 3). Initially, all 35 workers had the same function. We have measured their preferences concerning:

- The number of Morning-Late-Night shifts in a basic sequence.
- The number of days off after the last Morning, Late, and Night shift, respectively.
- A fixed weekly day-off.

We have used this input to compute the quality of a year roster for each person. Here we do not add the stand-by shifts; these are added later. Our solution approach is based on integer linear programming. Assuming that we have the complete set of possible, feasible rosters available, we can model the problem of finding the optimal set as an integer linear programming problem, in which we use a binary variable x_{js} to indicate whether person j gets a roster s , or not. The constraints in this ILP enforce the minimum occupancy and the fact that we can assign only one roster per person.

Since we do not know the full set of rosters, and since we cannot handle such a big ILP problem, we solve the LP-relaxation approximately using column generation, and we use this knowledge to find an approximate solution. To get the column generation going, we enumerate all (approximately 70.000) four-week rosters; this enumeration step is necessary to deal with the constraint that the workload per week should not vary too much. Moreover, we can easily compute the satisfaction that this roster provides to each of the employees, which enables us to remove the ones that do not score satisfactorily well. We use these four-week rosters to build our one-year roster. Ideally, we would be able to solve the pricing problem by solving a shortest-path problem in a layered graph, where each node in a layer models a four-week roster. This is computationally infeasible, though, and therefore, we solve the pricing problem for only two or three four-week periods at a time: given the ‘best’ partial solution for the first i four-week periods, we solve the pricing problem for the periods $i + 1, i + 2$, starting with the fixed roster for period i . We then add period $i + 1$ to the partial solution, and move one. In the meantime, we compute a number of additional rosters that we believe are worthy. After we have ‘solved’ the LP-relaxation, we solve the original ILP for the set of generated rosters.

3 Continuation

Before it could be run in practice, the problem got changed by a function differentiation: two groups of six people were split off. For the remaining 23 people, we can still apply the approach described above, but it does not work for the groups of six. Column generation is known to work well in case there are many feasible solutions, and for the groups of six it is hard to just find a solution that satisfies the basic constraints. Therefore, we have just started a constraint satisfaction approach.

Timetabling at German Secondary Schools: Tabu Search versus Constraint Programming

Frank Jacobsen, Andreas Bortfeldt and Hermann Gehring

University of Hagen, Germany

Frank.Jacobsen@gmx.de

1 Introduction

In recent years many solution approaches for different school timetabling problems (TTP) have been tried, among them tabu search, simulated annealing, genetic algorithms, and constraint programming [8]. However, most of the solution methods developed so far have been tested by means of only few (mostly only a handful) problem instances [6]. Further, until today there are nearly no reports on tests which subject different methods to a comparative analysis on the basis of the same benchmark instances and thus provide empirical evidence for the (relative) suitability of solution approaches.

In this paper, a tabu search algorithm [4] for timetabling at German secondary schools of the Gymnasium type is presented. The TSA is subjected to an extensive test including 1500 problem instances. The instances have been introduced by MARTE [6] and used for the test of his constraint programming [5] method. Therefore, the results obtained with the TSA and the CP method are finally compared here.

2 Problem description

German secondary schools can be compared with the British grammar schools, but they admit extensive choices to pupils [3]. Similar to MARTE [6], it is assumed here that the timetabling is essentially based on the following conditions:

- The lessons are given on the grade levels 5 to 13.
- The sets of teachers, rooms, and classes are fixed.
- The rooms are classified according to certain room types (e.g. gym hall).
- On each grade level one or (from level 7 onward) more teaching programs are offered. A program is fixing a selection of subjects and the number of weekly lessons for each subject. Therefore, there exist one or more pupil groups (PG) per class with the same teaching program for the pupils of a group. A typical feature of a teaching program is a choice of foreign languages and/or a study direction such as Social Sciences or Natural Sciences.
- The complete weekly teaching program is now specified as a set of lesson requirements (LR). A LR is a combination of one teacher, one subject, one or more PG's from one or more classes, and one room type. The teaching of a LR lasts always for a period (of 45 minutes). For each weekday the number of periods which are available for the timetabling is fixed.

The TTP of a secondary school of the Gymnasium type (GYM-TTP) combines the tasks of room and period assignment for pre-defined LR's and can be formulated as

follows. Assign a room of appropriate type and a period to each of the LR's in such a way that the following constraints are met:

- **Clash constraints** A1-A3: The scheduling of teachers, rooms, and classes or pupil groups, respectively, must avoid clashes.
- **Availability constraints** B1-B3: Teachers, rooms, and classes must be scheduled within their availability time windows.
- **Coupling constraints:**
 - C1 Certain LR's are to be scheduled for the same period.
 - C2 Certain pairs of congruent LR's are to be scheduled for two consecutive periods, and the same room is to be assigned to both of the LR's (2-hour lesson).
- **Distribution constraints:**
 - D1 The timetable of each class shouldn't contain idle periods.
 - D2 For each class the lessons should end as early as possible on each day.
 - D3 Certain LR's are to be scheduled for pre-determined periods.
 - D4 For teachers and for pupils the daily minimum and maximum number of lessons should be respected. Further, a lower and an upper limit of working days per week are to be considered for each teacher.
 - D5 For each class the daily minimum and maximum number of lessons on the same subject should be respected.

3 Mathematical model

The GYM-TTP is formulated as a binary optimization model. Except for D1 and D2, all constraints are categorized as hard.

Depending on the type of constraint, either LR's or so-called complex lesson requirements (CLR's) serve as the basis for the modelling. A CLR includes all LR's which, according to C1, are to be held at the same time. Further, both of the LR's of a 2-hour lesson are, according to C2, always assigned to the same CLR. A CLR must comprise either only 2-hour lessons or only (1-hour) LR's.

Two sets of binary decision variables – x_{np} and y_{mr} – are introduced. A variable x_{np} has the value 1 if the CLR n is scheduled for period p , and a variable y_{mr} has the value 1 if the LR m is scheduled for room r . Due to the planning of periods on the level of CLR's, the constraints C1 and C2 are automatically met. The remaining hard constraints are modelled explicitly.

The constraints D1 and D2 are integrated in the objective function f which is defined as $f = f_{ip} + f_{ef}$ and to be minimized. The term f_{ip} is summing up the number of idle periods over all classes, i.e. the unplanned periods which are, however, followed by lessons. The term f_{ef} measures the compactness of a timetable and, roughly expressed, sums up all variables x_{np} which are weighted with the indices of the periods of a day. For evaluation purposes and by means of an obvious lower bound lbf_{ef} , the compactness index is defined as $lbf_{ef}/f_{ef} * 100$.

4 The Tabu Search Algorithm

In the following, the essential properties of the proposed TSA, called TS-Gym, are described.

TS-Gym is a purely deterministic method. In the interest of a high robustness, stochastic components have been omitted.

A generated solution is represented by two vectors, a period vector and a room vector. The period vector is assigning a period to each CLR, while the room vector assigns a room of appropriate type to each LR. The search space contains feasible solutions, which meet all hard constraints (cf. section 3), and infeasible solutions as well.

An initial solution is generated by means of a specific construction heuristic. The heuristic is based on the sorting of the CLR's according to the difficulties arising with their scheduling, is using a graph coloring algorithm [1], and aims primarily at the generation of a feasible solution.

In TS-Gym two types of neighbourhoods are alternatively applied. In the case of the period-neighbourhood, a neighbour s' of a current solution s is derived through the assignment of a deviating period to exactly one CLR. In the case of the room-neighbourhood a neighbour s' of s results from a deviating assignment of a room for exactly one LR. The room-neighbourhood is only applied in situations where the current solution s violates one of the room constraints, A2 or B2. For both neighbourhood types, the best neighbourhood solution is determined by means of a specific evaluation function. The function is attaching a high weight to the violation of hard constraints and a low weight to the value of the objective function, f .

A best neighbourhood solution is accepted as the new best solution only if, in comparison to the current best solution, the number of violated hard constraints is reduced or, for the same number of violated constraints, the value of the objective function is improved.

The tabu list management is designed similar to DESEF et al. [2]. As in the latter case, two tabu lists are kept. The move list contains the (inverse) moves carried out recently. Purpose of the frequency list is to avoid too frequent shifts of individual CLR's. According to the aspiration by objective, the improvement of the best solution in the sense defined above is used as aspiration criterion.

5 Results and comparison with constraint programming

TS-Gym has been tested on a standard notebook (1.6 GHz Pentium-M, 1 GB RAM) using the 1500 test instances from MARTE [6][7] and a fixed parameter setting. The 1500 instances are subdivided into 6 test cases R1 to R6 which correspond to six secondary schools of various kinds and contain 250 instances each. The results obtained with TS-Gym and the CP method from MARTE are shown in Table 1.

For almost all instances both of the methods calculate a feasible solution, i.e., they achieve nearly the same high solution quality with respect to the share of instances solved to feasibility. It should be emphasized, however, that in the case of TS-Gym the consideration of the hard constraints D4 and D5 has not yet been implemented and therefore not been included in the test.

Table 1. Results for the secondary schools R1 to R6 [6][7].

Evaluation criterion	Method	R1	R2	R3	R4	R5	R6
Share of instances solved to feasibility (in %)	MARTE	100	97	100	98	99	92
	TS-Gym	100	96.4	99.6	98	99.2	92.4
Mean no. of idle periods	TS-Gym	11.6	8.4	8.4	7.0	7.3	7.3
Mean compactness	TS-Gym	65.8	73.1	73.3	77.3	76.0	76.7
Mean CPU time (in s)	TS-Gym	62.1	60.1	90.0	61.1	56.6	100.3

For the criteria "number of idle periods" and "compactness" comparative values are not available. Since the teaching program on the upper grade levels is similar to that of a university, where only moderate compactness requirements are to be met, these results and the computing times as well seem to be satisfactory.

Apart from the implementation of the constraints D4 and D5, the improvement of the parameterization and of selected components of TS-Gym will be the subject of further research.

References

1. Coleman, T. F. and Moré, J. J. (1983): Estimation of Sparse Jacobian Matrices and Graph Coloring Problems. In: SIAM Journal of Numerical Analysis, Vol 20, pp 187-209.
2. Desef, T., Bortfeldt, A. and Gehring, H. (2004): A Tabu Search Algorithm for Solving the Timetabling-Problem for German Primary Schools. In: Burke, E. K., Trick, M. (Eds): Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004), pp. 465-470.
3. Drexl, A. and Salewski, F. (1997): Distribution Requirements and Compactness Constraints in School Timetabling. In: European Journal of Operations Research, Vol 102(1), pp 193-214.
4. Glover, F. and Laguna, M. (1993): Tabu-Search. In: Reeves, C. R. (Ed): Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publications, Oxford etc.
5. Jaffar, J. and Maher, M. J. (1994): Constraint Logic Programming: A Survey. In: Journal of Logic Programming, Vol 19/20, pp 503-581.
6. Marte, M. (2002): Models and Algorithms for School Timetabling – A Constraint Programming Approach. Doctoral Dissertation at the Faculty of Mathematics, Computer Science and Statistics of the Ludwig-Maximilian-University of Munich, Munich.
7. Marte, M. (2004): Towards Constraint-Based School Timetabling. In: Hnich, B., Walsh, T. (Eds): Proceedings of the Workshop on Modelling and Solving Problems with Constraints, held at ECAI 2004, pp 140-154.
8. Schaefer, A. (1999): A Survey of Automated Timetabling. In: Artificial Intelligence Review, Vol 13, pp 87-127.

A Constructive Heuristic for the Travelling Tournament Problem

Graham Kendall¹, Wim Miserez², and Greet Vanden Berghe²

¹ University of Nottingham, School of Computer Science and IT, Jubilee Campus,
Nottingham NG8 1BB, UK
`gxk@cs.nott.ac.uk`

² KaHo St.-Lieven, Information Technology,
Gebr. Desmetstraat 1, 9000 Gent, Belgium
`Wim.Miserez@student.kahosl.be, Greet.VandenBerghe@kahosl.be`

Abstract. We present a two-stage heuristic for the travelling tournament problem. It consists of first creating patterns for good quality solutions for each team, after which these partial solutions are used/combined in an improvement heuristic to find overall solutions.

1 Introduction

The travelling tournament problem (TTP) involves minimising the total distance travelled when organising a double round robin tournament among a number of teams. Feasible solutions require that

- home and away games between two teams are not in consecutive timeslots (no repeaters),
- teams do not play more than three consecutive home/away games [7].

Sports scheduling has been the subject of many academic publications (e.g. [5, 6, 8, 9, 11–14]). Agnostonopoulos et al. [1] present the current best results for the problems presented on the TTP website (<http://mat.gsia.cmu.edu/TOURN/>). Instead of tackling the entire tournament as one problem, we propose a two-stage approach in which we solve the problem for individual teams first. Similar decomposition techniques have been successfully applied to other timetabling domains (e.g. to nurse rostering [2–4] and university timetabling [10]). The main motivation for decomposing the problem is that it considerably reduces the search space, and thus the computation time.

The method proposed in this abstract includes solving the corresponding Travelling Salesman Problem (Section 2.1) and generating feasible patterns with that solution (Section 2.2).

2 Constructive Heuristic

2.1 Travelling Salesman Problem

The total travelling distance for a tournament equals the sum of the travelling distances for each team. We decompose the problem into sub problems for each

team. Each of these sub problems involves searching for the minimum travelling distance for a team. If we ignore the constraint on consecutive home and away games, the problem equals a Travelling Salesman Problem (TSP) for which the cities correspond to the teams in the TTP. Each team should solve exactly the same TSP problem.

The best solutions for the TSPs corresponding to the different TTPs are presented in Table 1. The sequence of numbers denotes a sequence of optimal distance, in which the numbers refer to the order in which the teams appear in the distance matrix on the TTP website referred to in Section 1.

n	Dist.	TTP Solution
4	2011	1-3-2-4
6	2971	5-1-6-4-2-3
8	3480	6-4-2-3-5-1-7-8
10	3834	6-4-2-3-5-1-9-10-8-7
12	5600	7-6-3-2-4-8-10-9-12-11-5-1
14	7427	1-5-11-14-13-12-9-10-8-4-2-3-6-7
16	7443	7-6-3-2-4-8-10-9-12-13-15-14-16-11-5-1
18	7426	9-11-15-13-16-14-18-12-2-10-7-4-17-1-3-6-8-5
20	7426	19-7-20-4-17-1-3-6-8-5-9-11-15-13-16-14-18-12-2-10
22	7989	9-11-10-2-12-18-14-16-13-15-22-21-8-3-1-4-17-20-7-19-6-5
24	8091	15-22-24-21-23-8-3-1-17-4-20-7-19-6-5-9-11-10-2-12-18-14-16-13
26	8274	17-4-1-3-6-8-23-5-11-9-21-24-22-15-13-16-14-18-12-2-26-10-25-19-7-20
28	8393	11-25-19-7-20-4-17-1-3-6-8-23-5-9-21-24-22-15-13-16-14-18-12-28-26-2-10-27
30	9393	9-5-11-30-15-13-29-16-14-18-12-28-26-2-10-27-25-19-7-20-17-4-1-3-6-8-23-24-22-21
32	9412	7-19-25-27-10-2-26-28-12-18-31-14-16-32-29-13-15-30-11-5-9-21-22-24-23-8-6-3-1-4-17-20

Table 1. Solutions for the TSP that corresponds to the TTP of size n

2.2 Feasible Patterns

After the shortest tour has been calculated, we apply a heuristic step that modifies the optimal solutions into feasible patterns. We therefore introduce a number of home-away edges in the optimal tour, in such a way that no more than three consecutive home/away games occur. We call the results patterns, since they do not specify any ‘home’ opponent of the team yet. An example of a pattern for team 3 is presented in Fig. 1. The top row presents the shortest tour for the TSP, starting from city/team 3 (which is the first and the last node in the bottom row). The shortest tour is interrupted with additional away/home and

home/away edges: $5 \rightarrow 3$, $3 \rightarrow 1$, $6 \rightarrow 3$ and $3 \rightarrow 2$ in the example. The pattern denotes a feasible solution for team 3, provided that the assignment of home opponents does not introduce any repeaters. Obviously, the presented pattern is not optimal since it contains more home-away edges than necessary.

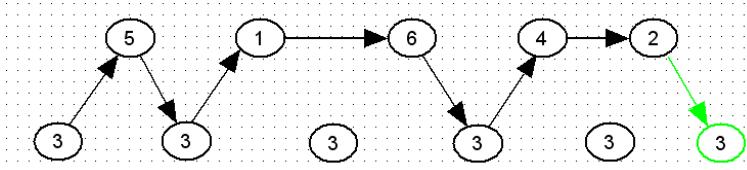


Fig. 1. Example of a pattern

In order to create the patterns, we first introduce a pattern array of length n . The elements in the array are 0 and 1. A 0 in the pattern means that the next team in the shortest path sequence should be visited without going home. A 1, on the contrary, means that the team should first go home before visiting the next team in the shortest path sequence. We do not put consecutive 1s in the patterns since the home games do not influence the travelling distance for the home team. The pattern corresponding to Fig. 1, for example, is 1001001. Any team always ends the competition at home, and thus the last element of the pattern array is always 1. This leaves $n - 2$ 0/1 elements to assign. The total number of possible assignments thus equals $2^{(n-2)}$. Table 2 indicates the number of feasible patterns (i.e. patterns representing solutions which have not more than 3 consecutive away games) for each of the TTP problems.

When mapping the possible patterns onto the shortest path, we can calculate the distance for each team-pattern combination. Although the shortest tours are equal for each team, the patterns are different. Those with the smallest distance are the most interesting patterns to select for the tournament. Obviously, the order in which home sequences or away sequences appear in a solution has no influence on the distance of the home team.

3 Optimising Heuristic

From the set of patterns, we select and combine those that lead to good quality solutions. In the optimisation step we improve the solutions with local search. Examples of the neighbourhood moves considered are:

- swap the first and the last opponent team in a sequence of three,
- swap entire sequences of opponent games,
- etc.

The results will be presented at the conference.

n	# feasible patterns	Best Dist.	Best feasible pattern
4	4	2134	0001
6	12	4065	010001
8	40	3727	00010001
10	137	4692	0001010001
12	463	6930	010010010001
14	1560	9986	01000100010001
16	5264	11810	01001000100010001
18	17945	10388	010001000100010001
20	60708	11480	01000101000100010001
22	205372	12198	0100010001001000100011
24	694769	14428	000100010001000100010001
26	2350385	15704	00100010001000100010001001
28	7951296	14192	0001000100011000100010010001
30	26899040	15475	010010001001000100010001001001
32	90998801	20377	010001000100010001000100010001

Table 2. Number of feasible patterns per problem of size n , the best feasible distance and an example of a pattern corresponding to that distance, when mapped upon the TTP solution

References

1. A. Anagnostopoulos, L. Michel, P. van Hentenryck and Y. Vergados: A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 2006 (to appear).
2. U. Aickelin, K. Dowsland: An Indirect Genetic Algorithm for a Nurse Scheduling Problem. *Journal of Computers & Operations Research*, 31(5), 2004, 761-778
3. J.F. Bard, H.W. Purnomo: Preference scheduling for nurses using column generation, *European Journal of Operational Research* 164, 2005, 510-534
4. P. Brucker, R. Qu, E.K. Burke, G. Post: A Decomposition, Construction and Post-processing Approach for a Specific Nurse Rostering Problem, proceedings of MIS-TA 2005: The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications, NY, USA, July 2005, 397-406
5. K. Easton, G. Nemhauser and M. Trick: Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach. In E. Burke and P. De Causmaecker (Eds). Selected papers from the 4th International Conference on the Practice and Theory of Automated Timetabling. LNCS 2740, 2003 100-109
6. K. Easton, G.L. Nemhauser and M. Trick: Sports scheduling, in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Ch. 52, J.T. Leung (Ed.) 2004, 1-19. CRC Press
7. K. Easton, G. Nemhauser and M. Trick: The traveling tournament problem: Description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol. 2239, Springer, 2001, 580-584
8. M. Henz, T. Mueller, and S. Thiel: Global constraints for round robin tournament scheduling. *European Journal of Operational Research*, Vol. 153, 2004, 92-101

9. M. Henz: Scheduling a major college basketball conference - revisited. *Operations Research*, 49, 2001, 163-168
10. J.H. Kingston: A Tiling Algorithm for High School Timetabling, Selected and Revised Papers of the 5th International Conference on Practice and Theory of Automated Timetabling, (PATAT 2004), Pittsburgh, Springer LNCS 3616, 2005, 208- 225
11. G.L. Nemhauser and M.A. Trick: Scheduling a major college basketball conference. *Operations Research*, Vol. 46, 1998, 1-8
12. J.A.M. Schreuder: Combinatorial Aspects of the Construction of Competition Dutch Professional Football Leagues. *Discrete Applied Mathematics*, Vol. 35, 301-312, 1992
13. M.B. Wright: Scheduling English Cricket Umpires. *Journal of the Operational Research Society*, 42:447-452, 1991
14. J.T. Yang, H.D. Huang and J.T. Horng: Devising a cost effective baseball scheduling by evolutionary algorithms. In Proceedings of the 2002 Congress on Evolutionary Computation, 2002, 1660-1665

Computational Complexity Issues in University Interview Timetabling

Yuuki Kiyonari¹, Eiji Miyano^{1*}, and Shuichi Miyazaki^{2**}

¹Department of Systems Innovation and Informatics,
Kyushu Institute of Technology, Fukuoka 820-8502, Japan
`{kiyonari@theory., miyano@}ces.kyutech.ac.jp`

²Academic Center for Computing and Media Studies,
Kyoto University, Kyoto 606-8501, Japan
`shuichi@media.kyoto-u.ac.jp`

1 Introduction

In the Department of Intelligence Science and Technology, Graduate School of Informatics, Kyoto University, there are approximately 20 professors, and 40 graduate course students in each year. In the final year of the course, every student submits a research thesis, and presents his/her work in twenty minutes to obtain a master degree. Presentation meeting is scheduled for two days, and in general, all professors attend the meeting and listen to all students' presentation. For each student, three professors (usually from the same department) are assigned as a referee basically according to the presentation topic and professors' research fields, and it is mandatory for referees to attend and evaluate the assigned students' presentation.

This presentation meeting used to be scheduled in one room, and there have been no serious problems. However, because of the increased number of graduate students, it became difficult to hold the meeting in two days this year, and we adopted a parallel session using two rooms. Then, there arises two major restrictions: (1) Two students evaluated by the same referee must be assigned to different timeslots. (2) Professors want to minimize the number of movements between rooms. Restriction (1) is a hard constraint, and the problem requires to decide if a feasible schedule exists. It is easy to see that this problem can be solved in polynomial time, and hence in this paper we focus on restriction (2), which is a soft constraint: We want to find a feasible schedule which minimizes the total number of movements of all professors.

To understand the problem, consider the following small example: There are six students s_1 through s_6 and six professors p_1 through p_6 . The assignment of professors to students is illustrated in Fig. 1: Student s_1 is evaluated by two professors p_1 and p_2 , and so on. One example of the schedule, say C_1 , is illustrated in Fig. 2. In C_1 , students s_1 through s_3 , and students s_4 through s_6

* Supported in part by Scientific Research Grant, Ministry of Japan, 16092223

** Supported in part by Scientific Research Grant, Ministry of Japan, 16092215 and 17700015

are scheduled to rooms r_1 and r_2 , respectively, in this order of timeslots. Then the cost of p_1 in the schedule C_1 is 2 since p_1 has to move twice, from r_1 to r_2 and then from r_2 to r_1 . The costs of p_2 and p_3 are 1 and 0, respectively. As a result, the cost of the schedule C_1 is $2 + 1 + 0 + 1 + 0 + 0 = 4$.

Student	s_1	s_2	s_3	s_4	s_5	s_6
Assigned professors	p_1, p_2	p_2, p_3	p_1, p_4	p_4, p_5	p_1, p_6	p_2, p_6

Fig. 1. Example of referee-assignment

	r_1	r_2
t_1	s_1	s_4
t_2	s_2	s_5
t_3	s_3	s_6

	r_1	r_2
t_1	s_1	s_4
t_2	s_5	s_2
t_3	s_3	s_6

	r_1	r_2
t_1	s_3	s_6
t_2	s_1	s_4
t_3	s_2	s_5

C_1

C_2

C_3

Fig. 2. Schedules C_1 , C_2 and C_3

It seems hard to attack this problem directly, and hence, we will consider two restricted problems, denoted ROOM and ORDER. ROOM takes an initial feasible solution as an input, as well as an assignment of referees to students. We are allowed only to exchange the presentation rooms of a pair of students assigned to the same timeslot; so it is prohibited to change the assigned timeslots. For example, C_2 in Fig. 2 is one possible output of ROOM when C_1 in Fig. 2 is an input schedule. C_2 is the result of exchanging rooms of s_2 and s_5 , by which we can improve the cost of schedule to 3. The second problem, called ORDER, takes the same inputs as ROOM. It allows to exchange the timeslots of two pairs, but does not allow to change the assigned rooms. For example, C_3 in Fig. 2 is one possible output of ORDER when C_1 is an input. Recall that a feasible solution can be found in polynomial time as mentioned above, and hence, allowing to give an initial schedule as an input is reasonable.

Our Contribution. In this paper, we investigate the time complexity of ROOM and ORDER. It is reasonable to assume that the number of students each professor evaluates, and the number of professors assigned to each student are bounded by, say, s and t , respectively. Problem $\text{ROOM}(s, t)$ is Room whose input is restricted as above, and problem $\text{ORDER}(s, t)$ is defined similarly. By definition, $\text{ROOM}(1, t)$ is trivially in \mathcal{P} for any t . This paper shows that (i) $\text{ROOM}(2, 1)$ is also polynomial-time solvable, but (ii) $\text{ROOM}(s, t)$ is generally \mathcal{NP} -hard and furthermore it remains intractable even if $s = 2$ and $t = 2$. As for the complex-

ity of ORDER, it is easy to see that ORDER(s, t) is polynomial-time solvable for $s \leq 2$. We show that (iii) ORDER(3, 1) is in \mathcal{P} .

Related Work. In educational timetabling, a set of resources such as teachers, students, rooms, and lectures must be assigned to a set of timeslots subject to certain hard and soft constraints. There are three main categories in educational timetabling, namely, school (or class-teacher), university course, and exam timetabling (e.g., see [7]). There are a large number of researchers investigating in detail the complexity of university timetabling [2–6, 8].

The interview timetabling problem treated in this paper can be regarded as the classical examination timetabling problem by considering students and referees in the former problem as exams and students in the latter problem, respectively. However, interesting parameter setting where we can derive a boundary between \mathcal{P} and \mathcal{NP} -hard is the case when s and t are small. For such settings, it is natural to interpret the problem as the interview timetabling rather than examination timetabling.

2 Complexity of ROOM

It would be trivial that ROOM(1, t) is in \mathcal{P} for any t because no professor needs to move and hence the cost is 0 for any schedule. However, the precise complexity of ROOM(s, t) for $s \geq 2$ is not evident. First, we consider ROOM(2, 1) and give a polynomial-time algorithm that solves it.

Let us call two students who are assigned to the same timeslot by an input schedule a *student-pair* (or simply, *a pair*). In the following, if we write a student-pair as (s_i, s_j) , it means that s_i and s_j are assigned to rooms r_1 and r_2 , respectively, by the current schedule. By “*flip a student-pair* (s_i, s_j) ”, we mean to exchange the rooms of s_i and s_j , namely, we change (s_i, s_j) to (s_j, s_i) . Without loss of generality, we assume that each student is evaluated by exactly one professor (namely, there is no student to whom no referee is assigned). So, if professor p is assigned to student s , we write $A(s) = p$ for convenience.

Starting from an initial schedule C , our algorithm decides, for each student-pair, whether to flip it or not, in a sequential manner. We first select an arbitrary pair, say (u, v) , and fix the rooms of this pair as it is, and focus on the student u . We select a pair (x, y) (if any) such that either x or y is evaluated by $A(u)$. If it is x , namely $A(u) = A(x)$, we keep the rooms of x and y as it is, so that the professor $A(u)$ does not have to move. If it is y , then we flip the pair (x, y) , again, so that $A(u)$ need not move. In this way, we continue determining the rooms of pairs, so that a professor in question does not have to move. When there is no pair to select, then we focus on the other student v of the initial pair, and do the same operation starting from v . If there is no pair to select, we close this “chain”, and start the next phase by selecting an initial pair again. The algorithm stops when all pairs are processed.

Theorem 1. ROOM(2, 1) is in \mathcal{P} .

Proof (Sketch). Clearly, the time complexity is polynomial. It is not hard to see that each phase gives rise to the cost of at most one, and if it is one, then the set of pairs selected in that phase causes the cost of one in any schedule. Hence the schedule the above algorithm outputs is optimal. \square

Next, we show an intractable case of $\text{ROOM}(s, t)$.

Theorem 2. $\text{ROOM}(s, t)$ for $s \geq 2, t \geq 2$ is \mathcal{NP} -hard.

Proof (Sketch). Consider the following problem MAX E2LIN2(3): We are given n variables x_1, x_2, \dots, x_n and m equations each with exactly two variables, $x_{i_1} \oplus x_{i_2} = a_i$ ($1 \leq i \leq m$, $a_i \in \{0, 1\}$) such that each variable appears at most three times in the equations. We are asked to assign 0 or 1 to variables so that the number of satisfied equations is maximized. It is known that MAX E2LIN2(3) is \mathcal{NP} -hard [1].

Given an instance I of MAX E2LIN2(3) with n variables and m equations, we construct an instance I' of $\text{ROOM}(2, 2)$. For each variable x_i ($1 \leq i \leq n$), we create a student-pair $(s_{i,1}, s_{i,2})$, and for each equation $e_j : x_{j_1} \oplus x_{j_2} = a_j$ ($1 \leq j \leq m$), we create a professor p_j . Referee-assignment is constructed as follows. Consider the j -th equation e_j ($1 \leq j \leq m$). If it is of the form $x_{j_1} \oplus x_{j_2} = 0$, then either (a1) assign p_j to $s_{j_1,1}$ and $s_{j_2,1}$, or (a2) assign p_j to $s_{j_1,2}$ and $s_{j_2,2}$. If $x_{j_1} \oplus x_{j_2} = 1$, then either (b1) assign p_j to $s_{j_1,1}$ and $s_{j_2,2}$, or (b2) assign p_j to $s_{j_1,2}$ and $s_{j_2,1}$. Observe that each professor appears twice in I' since each equation contains two variables, but three referees may be assigned to one student since a variable can appear three times. Hence, at this moment, a constructed instance is of $\text{ROOM}(2, 3)$. However, we can create an instance of $\text{ROOM}(2, 2)$ by appropriately choosing (a1) or (a2) ((b1) or (b2)), although we omit describing how to do it.

An assignment C for I naturally corresponds to a schedule C' of I' : If $x_i = 0$, then the rooms of $(s_{i,1}, s_{i,2})$ is the same as in the initial schedule. If $x_i = 1$, the rooms of $(s_{i,1}, s_{i,2})$ is flipped to $(s_{i,2}, s_{i,1})$. It is not hard to see that the number of unsatisfied equations under C is equal to the total number of movements of professors under C' . \square

3 Complexity of ORDER

Recall that the operation we are allowed in this problem is only to decide the timeslot of student-pairs. Hence, as the simplest example, if each professor judges at most two students, any exchange operation of timeslots of two student-pairs does not change the cost of the schedule. It follows that $\text{ORDER}(s, t)$ is in \mathcal{P} for $s \leq 2$ and any t since any solution is optimal.

In this section we present a polynomial-time algorithm to find an optimal solution for $\text{ORDER}(3, 1)$. Let $\text{cost}(C, p)$ be the number of movements of professor p under a schedule C . Note that if a professor p appears at most twice in an input referee-assignment, $\text{cost}(C, p)$ is the same for any schedule C as mentioned above. Even if p appears three times, $\text{cost}(C, p) = 0$ for any C if all three students

are assigned to the same room by the input schedule. These professors are called *non-potential* professors. If p appears three times, and if two of his/her students are assigned to the same room, and the other one is assigned to the other room, his/her cost can be one or two depending on the schedule. We call these professors *potential* professors. As in the case of $\text{ROOM}(2,1)$, let $A(s)$ denote the referee assigned to student s .

As before, we sequentially determine the schedule of each pair. We construct several blocks of student-pairs. Starting from an arbitrary initial student-pair (u, v) , we select a student-pair (x, y) where $A(u) = A(x)$ and $A(u)$ is a potential professor, if any. We schedule (x, y) to the timeslot next to (u, v) , so that two students professor $A(u)$ evaluates are assigned to continuous timeslots and to the same room. Next, we select a pair (w, z) such that $A(y) = A(z)$ and $A(y)$ is a potential professor, if any, and schedule (w, z) to the timeslot next to (x, y) , and so on. When there is no pair to select, we then go back to (u, v) , and perform the same operation starting from $A(v)$. This time, we schedule new pairs to *previous* timeslots of (u, v) . When there is no pair to select, the work on the current block is finished, and we start to construct a new block by selecting an arbitrary initial student-pair. Finally, blocks are scheduled in an arbitrary order.

Theorem 3. $\text{ORDER}(3, 1)$ is in \mathcal{P} . (Proof is omitted.)

4 Concluding Remarks

In this paper, we considered the time complexity of $\text{ROOM}(s, t)$ and $\text{ORDER}(s, t)$ for several values of s and t . The apparent next step in this research is to investigate the complexity of $\text{ROOM}(3, 1)$ and $\text{ORDER}(4, 1)$. An interesting generalization is to allow both operations of ROOM and ORDER simultaneously. The goal in this line is to consider the most general problem, namely, the problem without an initial schedule in an input.

References

1. P. Berman and M. Karpinski, “Improved approximation lower bounds on small occurrence optimization,” ECCC Report, TR03-008, 2003.
2. M. W. Carter and C. A. Tovey, “When is the classroom assignment problem hard?” *Operations Research*, Vol.40, No.1, pp.28–39, 1992.
3. E. Cheng, S. Kruk, and M. J. Lipman, “Flow formulations for the student scheduling problem,” in *Proc. 4th PATAT 2002*, LNCS 2740, pp.299–309, 2003.
4. T. B. Cooper, J. H. Kingston, “The complexity of timetable construction problems,” *Proc. 1st PATAT 1995*, LNCS 1153, pp. 283–295, 1996.
5. H. M. M. ten Eikelder, and R.J. Willemen, “Some complexity aspects of secondary school timetabling problems,” in *Proc. 3rd PATAT 2000*, LNCS 2079, pp.18–27, 2001.
6. S. Even, A. Itai, and A. Shamir, “On the complexity of timetabling and multi-commodity flow problems,” *SIAM J. Computing*, Vol.5, No.4, pp.691–703, 1976.

7. A. Schaefer, "A survey of automated timetabling," *Artificial Intelligence Review*, Vol.13, No.2, pp.87–127, 1999.
8. D. de Werra, "Some Combinatorial Models for Course Scheduling," *Proc. 1st PATAT 1995*, LNCS 1153, pp.296–308, 1996.

Local Search Heuristics for the Teacher/Class Timetabling Problem *

Yuri Kochetov¹, Polina Obuhovskaya² and Mikhail Paschenko¹

¹ Sobolev Institute of Mathematics, Russia

² Novosibirsk State University, Russia

obuhovskaya@ccfit.nsu.ru

1 Introduction

We consider the well known NP-hard teacher/class timetabling problem [1]. Variable neighborhood search and tabu search heuristics are developed to find near optimal solutions to this problem. The heuristics are based on two types of solution representation. For each of them we consider two families of neighborhoods. The first family uses swapping of time periods for teacher (class) timetable. The second family bases on the idea of large Lin–Kernighan neighborhoods. Computation results for difficult random test instances show high efficiency of the proposed approach.

2 Problem Formulation

In the teacher/class timetabling problem we are given the following finite sets: J is the set of subjects, K is the set of classes, L is the set of teachers, T is the set of time periods. These periods are distributed in 6 week days. By $T_l \subseteq T$ we denote the set of time periods which are available for teacher l . We suppose that classes are disjoint sets of students, students in a chosen class have the same subjects, and correspondence between subjects and teachers for a chosen class is one-to-one. The number of lessons per week for each class and each teacher is known in advance. We say that a timetable S is feasible if the following requirements are satisfied:

- a teacher l has at most one lesson at a time period t if $t \in T_l$ and no lessons otherwise;
- a class k has at most one lesson at a time period t ;
- each teacher must fulfill his (her) weekly number of lessons.

The objective function $F(S)$ is a penalty function for the following soft constraints:

- each teacher has no time gaps;
- each teacher has lessons in the most convenient time periods;
- each class has no double lessons.

* This work was partially supported by Russian Foundation of Basic Research, grant 04-07-900096

More exactly, we wish to minimize the following objective function:

$$F(S) = \sum_{l \in L} \sum_{d=1}^6 \alpha_{ld} f_{ld}^1(S) + \sum_{l \in L} \sum_{t \in T} \beta_{lt} f_{lt}^2(S) + \sum_{k \in K} \sum_{d=1}^6 \gamma_{kd} f_{kd}^3(S),$$

where positive α, β , and γ are the penalties and $f^i(S)$ is the number of violations of soft restriction i , $i = 1, 2, 3$. The optimization problem is NP-hard. Moreover, the decision problem on existence of a feasible solution is NP-complete. So, we introduce semifeasible solutions to enlarge the search space and apply metaheuristics for this space to find near optimal feasible solutions.

3 Solution Representations

We introduce two types of semifeasible solutions.

Definition 1. A timetable S^a is a semifeasible solution of the type a if it satisfies the restrictions b and c .

Definition 2. A timetable S^b is a semifeasible solution of the type b if it satisfies the restrictions a and c .

It is convenient to represent an arbitrary timetable S^a as a $\mathbf{K} \times \mathbf{T}$ matrix (S_{kt}^a) , $\mathbf{K} = |K|$, $\mathbf{T} = |T|$, with values in $\{0, 1, \dots, \mathbf{J}\}$, $\mathbf{J} = |J|$, where the k -th row is a timetable for the k -th class. Nonzero entries of the row mean subjects for the class k at the time period t ; $S_{kt}^a = 0$ means free time. In a similar way we represent S^b as a $\mathbf{L} \times \mathbf{T}$ matrix (S_{lt}^b) , $\mathbf{L} = |L|$, with values in $\{-1, 0, 1, \dots, \mathbf{K}\}$. Entries of the matrix mean classes for the teacher l at the time period t if $S_{lt}^b > 0$, and free time if $S_{lt}^b \leq 0$. The case $S_{lt}^b = -1$ means that the time period t is unavailable for the teacher l . The advantage of this representation is that it eliminates conflicts for teachers. The occurrence of conflicts in column happens when in a given period t more than one teacher is allocated to a class. A solution S^b is feasible if and only if each column has no conflicts. An arbitrary feasible solution S can be easily represented by (S_{kt}^a) and (S_{lt}^b) matrices. In order to evaluate the semifeasible solutions we introduce the following function

$$\bar{F}(S) = F(S) + \sum_{l \in L} \sum_{t \in T} \lambda_{lt} f_{lt}^4(S) + \sum_{k \in K} \sum_{t \in T} \mu_{kt} f_{kt}^5(S),$$

where $\min(\lambda, \mu) > \max(\alpha, \beta, \gamma)$ and $f^4(S)$, $f^5(S)$ are some penalty functions for the restrictions a and b . Obviously, $\bar{F}(S) = F(S)$ if S is a feasible solution. It is easy to realize a transition from a semifeasible solution S^a to S^b and back such that the number of positive items in $\bar{F}(S) - F(S)$ does not increase.

4 Neighborhoods

Now we introduce four families of neighborhoods:

- $N_i(S^a), i \geq 1$, denote swap neighborhoods of a semifeasible solution S^a ;
- $N_i(S^b), i \geq 1$, denote swap neighborhoods of a semifeasible solution S^b ;
- $LK_i(S^a), i > 1$, denote Lin–Kernighan neighborhoods of S^a ;
- $LK_i(S^b), i > 1$, denote Lin–Kernighan neighborhoods of S^b .

The neighborhood $N_1(S^a)$ consists of neighboring solutions which are obtained from S^a by swapping two different values of a given row in the matrix (S_{kt}^a) . Each element in this neighborhood is associated with a triplet $\langle k, t', t'' \rangle$, where t' and t'' are the time periods, k is the class, and $S_{kt'}^a$ and $S_{kt''}^a$ are the interchanged subjects. For $i > 1$, $N_i(S^a)$ are formed of solutions which are obtained by a sequence of interchanges with triplets $\{\langle k, t'_j, t''_j \rangle\}_{j \leq i}$, $k \in K$ is fixed. Families $N_i(S^b)$ are defined in a similar way. Moreover, only non-negative values of the matrix (S_{lt}^b) can be interchanged. We note that arbitrary feasible solution can be reached with the use of an appropriate sequence of neighboring solutions for the neighborhoods $N_1(S^a)$ or $N_1(S^b)$. A Lin–Kernighan neighborhood $LK_i(S^a)$ consists of i elements and can be described by the following steps [3].

1. Choose a triplet $\langle k, t', t'' \rangle$ such that the corresponding neighboring solution $S' \in N_1(S^a)$ is the best even if it is worse than S^a .
2. Put $S^a := S'$.
3. Repeat steps 1, 2 i times; if a triplet was used at steps 1 or 2 of previous iterations, it can not be used any more.

The sequence of triplets $\{\langle k_j, t'_j, t''_j \rangle\}_{j \leq i}$ defines i neighbors S_j of the solution S^a . We say that S^a is a local minimum with respect to the LK_i -neighborhood if $\overline{F}(S^a) \leq \overline{F}(S_j)$ for all $j \leq i$. A local minimum with respect to the LK_i -neighborhood is a local minimum with respect to N_1 and is not necessary a local minimum with respect to N_i , $i > 1$. Family $LK_i(S^b)$ is defined similarly.

5 Variable Neighborhood Search

We adjust the framework of the VNS metaheuristic [2] for our problem as follows.

1. *Initialization.* Find an initial semifeasible solution S ; choose a stopping condition and sizes of neighborhood families i_{max}, j_{max} .
2. *Repeat* the following sequence until the stopping condition is met:
 - (a) Set $i \leftarrow 1$; if $\overline{F}(S) = 0$ then STOP, return the optimal solution S .
 - (b) Repeat the following steps until $i = i_{max}$:
 - i. *Shaking.* Generate a solution S' at random from the $N_i(S)$.
 - ii. *Local search.* Use a local descent algorithm with respect to N_1 with S' as the initial solution; denote the obtained local minimum as S'' .
 - iii. *Move or not.* If $\overline{F}(S'') < \overline{F}(S)$ then put $S \leftarrow S''$ and goto 2(a); otherwise, set $i \leftarrow i + 1$.
 - (c) i. *Large neighborhood search.* Use a local descent algorithm with respect to neighborhood $LK_{j_{max}}$ with S as the initial solution; denote the obtained local minimum as S'' .

- ii. Change representation. If $\bar{F}(S'') < \bar{F}(S)$ then put $S \leftarrow S''$, otherwise change the solution representation; goto 2(a).

At the initial step 1 we generate S by a polynomial time heuristic. It has T stages. At each stage we solve an assignment problem.

6 Computational Results

We test the VNS algorithm on random instances with $T = 6 \times 5$ and $\alpha = 1 + \alpha'$, $\beta = 3 + \beta'$, $\gamma = 5 + \gamma'$, $\lambda = 10 + \lambda'$, $\mu = 10 + \mu'$, where $\alpha', \beta', \gamma', \lambda', \mu'$ are random noise, $0 < \alpha', \beta', \gamma', \lambda', \mu' \ll 1$. This rule removes plateaus and improves the landscape for local search methods. Each class has T lessons. Each teacher $l \in L$ has $T_l/5$ inconvenient time periods. The VNS algorithm produces 50 **KT** moves from a solution to a neighboring one. Table 1 presents average values of the objective function for the best found solutions in 50 trials. Each row of the table corresponds to one instance. For all instances VNS finds feasible solutions in all trials. For comparison, we present the results for a tabu search

n	L	K	J	$\sum_l T_l$	TS	TSR	VNS	VNSR
1	14	6	83	210	32.9	27.7	30.3	27.0
2	16	8	120	336	21.6	15.7	17.9	15.1
3	23	12	195	552	82.3	67.5	65.5	64.2
4	31	13	207	558	70.1	66.9	67.4	66.3

Table 1. Average values of $\bar{F}(S)$

method with and without changing the solution representation (columns TSR and TS). Table 1 shows that change of the solution representation is a useful idea for both methods. We hope it may be successfully applied for other approaches as well.

References

1. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco, CA, (1979)
2. Hansen, P., Mladenović, N.: Developments of Variable Neighborhood Search. In Ribeiro, C.C., Hansen, P.(Eds): Essays and surveys in metaheuristics. Kluwer Academic Publishers, (2002)
3. Kochetov, Y., Alekseeva, E., Levanova, T., Loresh, M.: Large neighborhood local search for the p-median problem. Yugoslav Journal of Operations Research **15** (2005), Number 1, 53-63

Time windows and constraint boundaries for public transport scheduling

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

School of Computing, University of Leeds,

Leeds LS2 9JT, UK

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

Public transport driver scheduling is the problem of determining the composition of a set of driver shifts (a *schedule*) for a day's transport operation requiring coverage by drivers, while minimising the operational cost (and/or robustness) of the schedule [2]. A *relief opportunity* (RO) is a (time, location) pair where drivers can be relieved. In the case of rail driver scheduling, most relief opportunities occur when a train stops at a station. It is frequently the case that trains will stop for some time before continuing; this gives rise to *windows of relief opportunities* (WROs).

Driver scheduling models usually approximate windows of relief opportunities by their arrival time. WROs could be expanded into sets of 1-minute-apart ROs, but the resulting model is unsuitable to be solved using the generate-and-select (GaS) approach [1], because the number of valid shifts becomes unmanageable in size [3]. However, it is expected that not all of the ROs derived from WROs will be vital for yielding more efficient solutions. For example, some of these new ROs that are close together are likely to be redundant.

Applying the 1-minute expansion to a typical instance of the rail driver scheduling problem in the UK is likely to result in several hundred new ROs. The problem is then how to select which of these potential ROs to include in the expanded model; a brute-force approach (say, trying all subsets of size n , one at a time) is clearly unsuitable, even for a fixed number n of ROs. In this work we present a set of heuristics to select these ROs.

Many scheduling constraints can be looked at in terms of the *boundaries* they define. Figure 1 depicts such an example: given an RO r on vehicle v at time t , a maximum work spell length of x minutes will define a boundary in vehicle v at time $t - x$, such that any spell on vehicle v ending at r will satisfy this constraint if the spell starts at or after $t - x$, and will break the constraint otherwise. If $t - x$ falls inside a WRO w at vehicle v (but not at its arrival time), then considering relieving inside w at $t - x$ leads to forming a spell which was invalid on the simplified, relief-on-arrival model. This would indicate that the RO at vehicle v and time $t - x$ is a good candidate to be included in the extended model, because it allows for a new spell to be generated. There may be more than one potential RO within w that fall on or after $t - x$, and all of these could in principle be included in the expanded set, although this would probably result in too many ROs being selected.

We derive a general framework to look at scheduling constraints in terms of time boundaries. We show its application using different constraints, including



Fig. 1. Looking at boundary conditions for the *maximum spell length* constraint. A maximum spell length of x defines an interval $[t - x, t]$ for the start of a spell ending at time t . A new spell can be formed if the RO at time $t - x$ is added to the model.

the existence of a feasible travel link and the maximum spell length. In particular, we apply this analysis to the first constraint on a set of real-life driver scheduling instances from four different UK railway operations. By adding these selected ROs, we are able to improve on best-known solutions. A further study on one of these instances shows that the same result could have been achieved by adding just one of the about 90 ROs added by our heuristic. This reinforces our claim that a careful selection of the ROs to add may be crucial in achieving the best solutions.

Analysing a set of scheduling constraints simultaneously opens up a range of possible algorithms/heuristics. A straightforward way of doing so is to look at each constraint separately, and then somehow merge the sets of ROs obtained. A completely opposite approach is to observe that the structure of the new spells/shifts arising from the consideration of scheduling constraints is usually similar across different constraints, e.g. new spells are obtained by adding a piece of work at the start of a valid spell in the simplified model, and making the new spell start properly inside a WRO. Therefore, a possible algorithm consists in forming new spells/shifts with that common structure, then test whether these are valid and suggest new ROs to be considered.

We develop two such algorithms: one working at spell level; the other, at shift level, using the shifts created in the generation phase of a GaS solver as a base for creating new shifts. Results show that this kind of approach can effectively encompass several scheduling constraints simultaneously.

References

1. Sarah Fores, Les Proll, and Anthony Wren. TRACS II: a hybrid IP/heuristic driver scheduling system for public transport. *Journal of the Operational Research Society*, 53:1093–1100, 2002.
2. Raymond S. K. Kwan. Bus and train driver scheduling. In J. Y-T Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 51. CRC Press, 2004.
3. Ignacio Laplagne, Raymond S. K. Kwan, and Ann S. K. Kwan. A hybridised integer programming and local search method for robust train driver schedules planning. *Lecture Notes in Computer Science*, 3616:71–85, 2005.

Minimizing the Carry-Over Effects Value in a Round-Robin Tournament

Ryuhei Miyashiro¹ and Tomomi Matsui²

¹ 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

² Department of Information and System Engineering,
Faculty of Science and Engineering, Chuo University,
Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan
matsui@ise.chuo-u.ac.jp

In this abstract, we deal with the problem of minimizing the carry-over effects value in a round-robin tournament. The carry-over effects value is an index of quality of a round-robin tournament schedule. We propose an effective algorithm for generating schedules of small carry-over effects values. The proposed algorithm produces better schedules than the previous best ones in short computational time.

A round-robin tournament with the following properties is considered in this abstract:

- the number of teams (or players etc.), n , is even;
- the number of *slots*, i.e., the days when matches are held, is $n - 1$;
- each team plays one match in each slot;
- each team plays every other team once.

Suppose that, in a round-robin tournament of high-contact sports (such as rugby and American football), team 2 is very strong and another team will be exhausted after the match against team 2. In this situation, which is a better schedule, Figs. 1 or 2? In Fig. 1, five of seven opponents of team 1 play team 1 just after playing team 2. Accordingly, team 1 is considered to have much advantage due to team 2. On the other hand, in Fig. 2 each team (except team 2) derives the advantage from team 2 at most once. In this regard, the schedule of Fig. 2 is better than that of Fig. 1. Such quality of a schedule can be measured by the *carry-over effects value*. In the following, the definition of the carry-over effects value is introduced.

It is said that team i gives a *carry-over effect* to team j if a team plays i in slot s then j in slot $s + 1$ ($s \in \{1, 2, \dots, n - 1\}$; regard slot n as slot 1). For a given schedule, the *carry-over effects matrix* C (coe-matrix for short) is a non-negative matrix whose element c_{ij} denotes the number of carry-over effects given by team i to team j in the schedule. By its definition, every coe-matrix satisfies the following:

- the sum of each row is $n - 1$;
- the sum of each column is $n - 1$;
- every diagonal element is 0.

	1	2	3	4	5	6	7	
1	8	3	4	5	6	7	2	$\begin{pmatrix} 0 & 5 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 2 & 1 & 0 & 3 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 3 & 0 & 2 & 0 & 0 & 2 \\ 1 & 1 & 0 & 2 & 0 & 2 & 0 & 1 \\ 2 & 0 & 3 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 2 & 3 & 0 \end{pmatrix}$
2	3	4	5	6	7	8	1	
3	2	1	6	8	5	4	7	
4	5	2	1	7	8	3	6	
5	4	7	2	1	3	6	8	
6	7	8	3	2	1	5	4	
7	6	5	8	4	2	1	3	
8	1	6	7	3	4	2	5	

Fig. 1. Schedule whose coe-value is 140 and its coe-matrix

The *carry-over effects value* (coe-value for short) is defined as $\sum_{i,j} (c_{ij})^2$.

The *carry-over effects value minimization problem* is to find a schedule of which coe-value is minimum. Obviously, the coe-value of a schedule of n teams attains the lower bound $n(n - 1)$ when all non-diagonal elements of the corresponding coe-matrix are 1. Such schedules are called *balanced*. In a balanced schedule, carry-over effects spread as evenly as possible. Fig. 2 shows a balanced schedule for $n = 8$ and its coe-value is $n(n - 1) = 56$, while the coe-value of Fig. 1 is 140.

Russell [2] proposed the carry-over effects value minimization problem, and an algorithm for constructing a balanced schedule when n is a power of two. In addition, it is conjectured that there is no balanced schedule unless n is a power of two. This conjecture is still open; for $n = 6$ and 10, it was verified by computation.

Russell also proposed a constructive heuristic algorithm to obtain schedules of small coe-values for $n = p^m + 1$, where p is an odd prime and $m \geq 1$. (Note that when $n \leq 20$, every even n is either a power of two or the form $p^m + 1$.) The heuristic algorithm produces schedules whose coe-values are 60, 138, 196, 260, 428 and 520 for $n = 6, 10, 12, 14, 18$ and 20, respectively. For $n = 6$, the schedule by Russell is indeed optimal. However, for $n = 10$ and 12, better schedules were recently obtained. For $n = 10$, Trick [3] reported a schedule whose coe-value is 122; for $n = 12$, Henz, Müller and Thiel [1] did a schedule whose coe-value is 188 (see Table 1). Both of them used constraint programming, and with constraint programming it seems difficult to find better schedules for larger n in practical computational time.

In the following, we propose a simple heuristic algorithm, which quickly generates better schedules than the previous best ones for $n \geq 14$. Our algorithm exploits the *circle method* (or polygon method), a well-known algorithm for constructing a round-robin tournament schedule. However, it has not yet been discussed in the context of minimizing the carry-over effects value.

The algorithm of the circle method is as follows:

in slot s ($s \in \{1, 2, \dots, n - 1\}$),

- team n plays team s ;
- team i plays team j for $(i + j) \equiv 2s \pmod{n - 1}$.

	1	2	3	4	5	6	7
1	4	5	6	7	8	2	3
2	5	4	8	3	6	1	7
3	8	6	5	2	4	7	1
4	1	2	7	6	3	5	8
5	2	1	3	8	7	4	6
6	7	3	1	4	2	8	5
7	6	8	4	1	5	3	2
8	3	7	2	5	1	6	4

Fig. 2. Balanced schedule

	1	2	3	4	5	6	7
1	8	3	5	7	2	4	6
2	7	8	4	6	1	3	5
3	6	1	8	5	7	2	4
4	5	7	2	8	6	1	3
5	4	6	1	3	8	7	2
6	3	5	7	2	4	8	1
7	2	4	6	1	3	5	8
8	1	2	3	4	5	6	7

Fig. 3. Schedule with the circle method

In the rest of this abstract, we denote the schedule created with the circle method by \mathcal{C}_n , where n is the number of teams. In \mathcal{C}_n , on the assumption that playing itself means playing team n , every team except n plays matches in the following order or its cyclic permutation: $1, 3, \dots, n-1, 2, 4, \dots, n-2$ (see Fig. 3). Accordingly, the coe-value of \mathcal{C}_n is very large. (For instance, the coe-value of \mathcal{C}_{10} is 468. We conjecture that \mathcal{C}_n gives an optimal solution for *maximizing* the coe-value.)

Consider permuting of the columns, i.e. slots, of \mathcal{C}_n . For a permutation σ on the set of the slots $\{1, 2, \dots, n-1\}$, we construct the schedule $\mathcal{C}_n(\sigma)$ whose s -th column is the $\sigma(s)$ -th column of \mathcal{C}_n . In addition, we define the sequence $p(\sigma)$ as follows: $p(\sigma) = (\sigma(2) - \sigma(1), \sigma(3) - \sigma(2), \dots, \sigma(n-1) - \sigma(n-2), \sigma(1) - \sigma(n-1)) \bmod (n-1)$. For a permutation σ , it is observed that if some elements of $p(\sigma)$ has a same value, particular elements of the coe-matrix of $\mathcal{C}_n(\sigma)$ increase; for instance, the schedule \mathcal{C}_n , which has a large coe-value, corresponds to the identical permutation and $p(\sigma) = (1, 1, \dots, 1)$. Thus, we expect that a good schedule is obtained by a permutation σ such that elements of $p(\sigma)$ are as different as possible.

To obtain schedules of small coe-values, we generated a number of permutations σ randomly. Our algorithm produced schedules whose coe-values are 254, 412 and 496 for $n = 14, 18$ and 20 respectively, in less than 1 second (CPU: Pentium III 1.0 GHz, RAM: 1024 MB); all of which are better than the previous results. The best coe-values after two days of random generation are 400 and 488 for $n = 18$ and 20, respectively (see Table 1).

Finally, it should be noted that we obtained schedules whose coe-values are 108 and 176 for $n = 10$ and 12, respectively (Table 1). These results were achieved by adding constraints to the constraint programming formulation proposed by Trick [3]. Due to the space limitation, the detail is omitted.

References

1. Henz, M., Müller, T., Thiel, S.: Global constraints for round robin tournament scheduling. European Journal of Operational Research **153** (2004) 92–101

Table 1. Upper bounds of the carry-over effects value

#teams	old best (status)	our results
4	12 ([2], balanced)	
6	60 ([2], optimal)	
8	56 ([2], balanced)	
10	122 ([3])	108
12	188 ([1])	176
14	260 ([2])	254
16	240 ([2], balanced)	
18	428 ([2])	400
20	520 ([2])	488

2. Russell, K.G.: Balancing carry-over effects in round robin tournaments. *Biometrika* **67** (1980) 127–131
3. Trick, M.A.: A schedule-then-break approach to sports timetabling. In: Burke, E., Erben, W. (eds.): Practice and Theory of Automated Timetabling III (PATAT 2000, Konstanz, Germany, August, selected revised papers). Lecture Notes in Computer Science, Vol. 2079. Springer-Verlag, Berlin Heidelberg New York (2001) 242–253

A Constraint Logic Programming Based Approach to the International Timetabling Competition

Patrick Pleass¹, Mark Wallace², Mauro Bampo³

^{1,2} Faculty of Information Technology

Monash University, Melbourne, Australia

{patrick.pleass, mark.wallace}@infotech.monash.edu.au

³ Faculty of Engineering

University of Bologna, Bologna, Italy

mbampo@gmail.com

This paper outlines the modeling, implementation and refinement of a solution to the International Timetabling Competition using Constraint Logic Programming methods. This is primarily carried out within the ECLiPSe constraint programming framework using lib(ic), the hybrid integer/real interval arithmetic constraint solver library.

The International Timetabling Competition, organized by the Metaheuristic Network and sponsored by PATAT (Practice and Theory of Automated Timetabling) was held in 2003. The competition presented a reduced university course timetabling problem and associated problem datasets designed by Ben Paechter. The aim of the competition problem was to deliver feasible timetables, in a set execution time that meets all hard constraints and minimized occurrences of soft constraints.

Although this competition has already been held and winners announced [1,2,3,4], the outcome has provided researchers with a number of independently verified solutions and performance measures using a variety of different approaches. Further to this the Center for Emergent Computing at Napier University have posted new "Harder" Instances for the University Course Timetabling Problem [5] that can further challenge heuristic development in this field.

The aim of this research is to provide a solution to the timetabling problem using as much as possible the ECLiPSe framework and minimal use of external custom-built metaheuristics and solvers. The performance of this approach is then compared to the competition results and differences analyzed and discussed. This approach introduces a number of design challenges in providing acceptable performance within ECLiPSe as opposed to a custom built heuristic. These challenges are outlined and discussed.

The approach followed consists of the following stages:

Modeling the problem data

Data from the input file format specified by the competition is loaded into the constraint engine as a set of atomic facts such as:

```
timeslot(timeslot_id)  
student(student_id,[classes]),
```

```
room(room_id,capacity,[available_features])
class(class_id,[required_features],TIME,ROOM)
```

The objective is to find TIME and ROOM for each defined class that meets all hard constraints and as many soft constraints as possible in the given time.

Modeling the hard constraints

The hard constraints to be implemented include:

- No student attends more than one event at the same time
- The room allocated to an event is big enough to house all students and meets all feature requirements
- Only one event is in each room for any timeslot

There are multiple ways to model these constraints within ECLiPSe. We present a high performance solution that minimizes the search domain to reduce the total search space for the next stage of the solution. Central to this step is the application of the `alldifferent(1)` predicate to an array of compound variables (ROOM and TIME) unified with the `element(3)` predicate.

Modeling the soft constraints

The soft constraints to be implemented are as follows:

- Minimize students with a class in last slot of each day
- Minimize students with two consecutive classes
- Minimize students with a single class on a single day

There are many strategies that may be employed within ECLiPSe to minimize these values. We employ in the first instance using selective constraint propagation techniques and then extend as performance dictates to other strategies based on constraint based local search.

Non-Exhaustive Search Strategies

As more variables are added to the problem, the search space grows exponentially, and left unchecked a CLP based system will search all possibilities. We devise heuristics that perform effective search strategies that focus on promising parts of the search tree in order to avoid an exhaustive search. This step will also utilize local and hybrid search and repair, chain swap and large neighborhood search.

Our approach is to find a complete assignment of variables that meet all hard constraints and as many soft constraints as possible in reasonable time. The total time the competition allows for finding a solution is 564 seconds on the hardware we employed. Reasonable time in our case is less than half of this time. The remaining time is used to perform constraint based local search to improve the solution.

Backtrack-Free Constructive Algorithms

To be able to produce solutions in reasonable time we have employed a backtrack-free, forward-checking constructive method as described by Schaefer [6]. This method will move from variable to variable and select the best value that meets the hard constraints and meets as many soft constraints as possible. The domains for each of the unassigned variables are then pruned to ensure that a total assignment of values exists before continuing.

The task is to find values for the time T and room R variables for each class. During the constructive search we focus only on the Time variable as most of the soft constraints rely on this. However with the assignment of every time variable, we also combine a channeling constraint that ensures that for any Time selection for a class there is at least one suitable room available that meets all hard constraints. The final allocation of rooms happens after the time allocation is complete.

References

1. Kostuch, P: The University Course Timetabling Problem with a 3-phase approach. PATAT 2004.
2. Jaumard, B., Cordeau, J., Morales, R.: Efficient Timetabling Solution with Tabu Search. International Timetable Competition, 2003.
3. Bykov, Y.: The Description of the Algorithm for International Timetabling Competition. International Timetable Competition, 2003.
4. Di Gaspero, L., Schaefer, A.: Timetabling Competition TTComp 2002: Solver Description. International Timetable Competition, 2003.
5. Lewis, R. and Paechter, B: New "Harder" Instances for the University Course Timetabling Problem <http://www.emergentcomputing.org/timetabling/harderinstances.htm>
6. Schaefer, Andrea: Combining Local Search and Look-Ahead for Scheduling Constraint Satisfaction Problems. Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), 1997.
7. Liatsos, V.: An Environment for a Resource Allocation Problem in CLP, MSc Thesis, University of London, 1995

Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search

Malek Rahoual¹, Rachid Saad²

¹ IBISC - FRE 2873 CNRS - Université d'Evry Val d'Essonne,
Tour Evry 2, 523 place des terrasses de l'Agora, 91000 EVRY, France.
mrahoual@lami.univ-evry.fr

² Département d'informatique, Université Mohamed Bougara, Boumerdès, Algeria.
rachid_saad2003@yahoo.com

Abstract. As demand for Education increases and diversifies, so does the difficulty of designing workable timetables for schools and academic institutions. Besides the intractability of the basic problem, there is an increasing variety of constraints that come into play. In this paper we present a hybrid of two metaheuristics (genetic algorithm and tabu search) to tackle the problem in its most general setting. Promising experimental results are shown.

Keywords: timetables, combinatorial optimization, genetic algorithms, tabu search, computational complexity.

1 Introduction

In academic institutions, nested groups of students (comprising streams, sections ...) are concerned by a set of subjects. A subject may be a lecture of some specific course or a tutoring or a lab, or any other meeting involving the group on a regular basis. For example, the lecture of the course entitled *MATH3802A* is a subject. Tutoring associated with the same course is another subject. Each subject takes a certain length of time whose unit is referred to as a '*period*'. Each subject may be broken down into a number of meetings to be scheduled.

To solve the timetabling problem (TTP) is to assign a qualified teacher to each subject and a time-slot together with a classroom of a suitable capacity and characteristics to each meeting. The assignment of times-slots, classrooms and teachers is subject to constraints that depend on the nature of the institution and its priorities. These constraints fall into three categories: physical constraints, which provide among others that no student can attend two different meetings at the same time; preference constraints and specification constraints bearing on some particular meetings, which, for example, must be held in some specified time window [3][4][5][8][10].

2 Application of genetic algorithms to timetabling problem

A Genetic Algorithm (GA) makes a population of individual solutions evolve under the control of two operators: ‘mutation’ and ‘crossover’. Mutation operates on one or possibly many genes (attributes) of a solution by altering their values. Crossover consists in mating the parental genes pairwise, yielding offspring with new properties. Only the best-fit individuals are likely to survive down the generations. We encoded our timetables as vectors associating 3 genes with each meeting, referring to the period, the classroom and the teacher assigned to the meeting. In a bid to promote a well-spread search of the space of solutions, we allowed our initial generation to be constructed randomly by assigning a random time-slot along with a teacher to each meeting in a “greedy” fashion. We then defined our objective function to be the weighted sum of all the violated constraints, each constraint being associated with a penalty (a weight) in proportion to the importance we ascribe to the constraint. The individuals (timetables) are ranked in the order of descending fitness conditions (as measured by the objective function), the best-fit individual being ranked 0. The probability of replacing an individual i is then defined to be the ratio of $rank_i$ to the sum of all the ranks. In other words, the poorer the fitness condition, the greater the probability of an individual being replaced.

2.1 Mutation

The simplest way to define a mutation operator is to select randomly both the gene and the gene value to which mutation is to be applied. For large sized instances, however, this may lead to unacceptable running times because of the poor choice of the genes. In contrast, our mutation operator operates on the most problematic genes. With each meeting m of M , an integer-valued variable is associated representing a violation record. The evaluation process consists in computing the penalty record incurred by the timetable: It adds all the violation records of meetings involved in any constraint of the timetable.

Thus, the violation record of a meeting is highest when the meeting is most problematic. The mutation operator then selects the gene to be muted as a function of the violation record. Likewise, a violation record is computed for each possible allele (gene value). The new value to be assigned to the mutating gene is deduced similarly as the sum of the penalties of all the constraints that will be violated if the change is accepted.

2.2 Dedicated mutations

We have designed a mutation operator for each of the three types of constraint. Each mutation operator of a given type maintains a violation record for that type for each meeting in M . This proved useful in many respects. First, the fact that only one class of constraint is handled in a mutation results in a significant gain in computing time. Secondly, in keeping separate records on different types of constraints we gain a better insight into which of the types is most violated, and by the same token, a means

is provided by which the sources of the problem can be accessed direct and dealt with. Thirdly, we now have another means (other than penalty value) to distinguish certain types of constraints and single them out as more important than others: it merely suffices to call their corresponding mutation operators more frequently.

2.3 Crossover

The role of crossover is to enable the combination of the good properties borne by the so-called parent-individuals. Our crossover operator is based on Abramson's encoding scheme [1][2], where a timetable is represented as a vector of lists, one for each time-slot. Each list brings together the set of meetings assigned to that time-slot. The crossover operator consists then in swapping groups of the two parents.

3 Hybridizing genetic algorithm with tabu search

Violation-driven Mutation (VDM) provides an effective means (time-wise) by which to intensify search [6][7]. However, the resulting tendencies to explore vs. exploit the search space often conflict with each other. The reason is that VDM may lead to premature convergence, wherein the search process gets stuck in a local optimum. Moreover, GAs are oblivious to the history of the search process, since only the population of individuals may be regarded as a short-term memory. On the other hand, we feel that Tabu Search (TS), with its numerous built-in strategies and features is better equipped to offset both shortcomings. For one thing, TS distinguishes itself by a greater ability to jump out from local optima thanks to its diversification strategy [6][7]. Furthermore, it makes use of both a short- and a long-term built-in memory to investigate the search space. Tabu Search proceeds by stepwise improvement. Thus, at each step of the algorithm, we move from one solution to another through an operation referred to as a 'move'. Attributes modified during a move become 'tabu' for a certain space of time in terms of the number of iterations. A list called Tabu list contains all tabu-attributes. Several hybridizing schemes are possible [9][12]. For one thing, two mutations on two identical individuals are most likely to perform the same selections of their genes. On the other hand, the mutation of an individual may bring it back to an already known state. Our solution requires that we keep two tabu-lists: a long-term list ("Long_list") and a short-term one ("Short-list"). "Long_list" is dedicated to storing the new values assigned to the genes so as to prevent new mutations from performing the same selection again. "Short_list" stores former gene values to prevent future mutations from restoring them.

4 Experimental tests

To analyze the performance of our algorithms, which were developed in C, we carried out an array of tests on instances of our own choosing, on benchmarks from literature

[2][11] as well as on a real case instance. We performed the tests using a micro-computer of type Pentium 4 (2.4 GHz, 512 Mo de RAM) operating on Linux 2.0.

The purpose of the first set of tests was to set the parameters of the algorithms. In the absence of relevant theoretical results on the subject, we had to resort to empirical tests to set our parameters. To determine each parameter value, we ran 10 tests per instance. Each such set of tests determined the most appropriate value of a given parameter, all else being unchanged. Crossover and mutation probabilities were set at 0,75 and 0,3 respectively. The number of iterations was set at 20000 and the population size at 300 individuals. As for the sizes of the tabu lists, the sizes of *Long_list* and *Short_list* were 7 and 5 respectively.

4.1. Theoretical instances

The tests were performed on 4 types of problems: problem 1 is constituted of 64 subjects with 12 teachers and 16 classrooms. Problem 2 has 100 subjects, 21 teachers and 25 classrooms. Problem 3 is made of 150 subjects, 26 teachers and 31 classrooms. Problem 4 has 200 subjects, 33 teachers and 37 classrooms. For each problem, we allowed the number of periods to vary between 20, 15 and 12 to test the efficiency of the algorithm for increasing period sizes. Run times are given in seconds.

The tests show that the use of a violation-driven mutation speeds up search while compromising the stability of search and the rate of success. The genetic algorithm hybridized with tabu search provides much better results in terms both of the quality of the solution and of the convergence rate, thanks to the tabu component of the algorithm which provides a better spread of the search. This is illustrated in the two charts above featuring a comparison between a hybrid GA and a classic GA; a comparison between a GA that uses plain mutation and a GA that uses a VDM; and a comparison between a GA using a period-based cross-over and one using plain crossover.

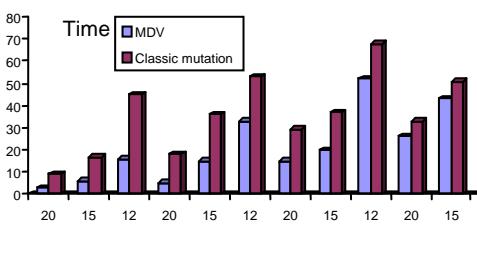


Fig. 1. Comparison between a classic GA and a GA with MDV.

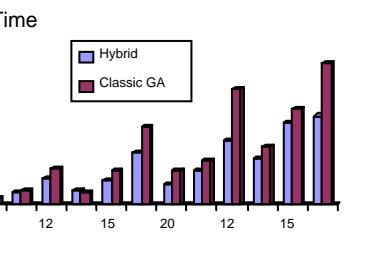


Fig. 2. Comparison between a hybrid GA and a classic GA.

4.2. Instances from literature

In our set of tests on instances from literature [2][11], we ran our algorithms on the challenging benchmarks Hdtt (*High difficult time-tabling*). These are difficult instances, because all the capacities are stretched thin: every teacher and every classroom must be assigned, and there must be no idle period at the optimum.

Using an IBM SP2, Abramson and Dang [2] have tested these instances with two metaheuristics: simulated annealing and tabu search. We performed 20 runs for each instance. There is a similarity between the results found in literature [2][11] and those provided by our algorithms. For larger instances (Hdtt7 and Hdtt8), more iterations are required in order for the quality of the solutions to be conclusive. A parallel version of the hybrid algorithm would be of much help as it will make it possible to deal with larger populations and more iterations.

4.3. Concrete Example

We have applied the hybrid algorithm to the TTP instance of the University of Science and Technology Houari Boumediene USTHB of Algiers, which has 500 groups, 1000 courses, 3000 teachers, about 5000 subjects to be scheduled on six consecutive periods a day for six days. We had two types of rooms to hold lectures and labs: 24 amphitheatres and 181 classrooms. Before the algorithm was applied to the ten faculties of the university, the department of Computer Science had tried it on its own Faculty. The manual designing of timetables in that Faculty used to take a tremendous amount of time (three to four weeks). With our algorithm, that time was reduced to just one hour in average.

Conclusions

Genetic Algorithms and Tabu Search provide a great flexibility of use when it comes to solving combinatorial problems. We exploited this flexibility to design algorithms capable of generating timetables for any type of academic institution, a problem intractable for approximation. Our algorithms process more than 11 constraints from among the most common ones, with the possible extension to others.

The idea of hybridizing of GA and TS stems from our desire to reap the benefits of both methods: the simplicity of use of GAs on the one hand, and such ability to jump out from local optima through a more diversified and balanced search as provided by TS, on the other hand. This hybridizing is enhanced by a host of ingredients of our own choosing and implementing. One of these is the so-called Violation Driven Mutation, which provides us with an intelligent operator capable of detecting and solving sources of conflicts. The idle time that might be generated in the process is eliminated via tabu-search.

The efficiency of this method makes itself felt on both counts of the convergence rate (running time) and the rate of success, as a result of a good compromise between

expansion and intensification, in terms of exploring new regions of the search space (through tabu-list) versus exploiting the solutions found (through crossover). In view of the variety of settings in which our algorithms fared well in comparison with other algorithms from literature, we conclude that our method together with its ingredients is pertinent to the timetabling problem.

As an extension to this work, we contemplate parallelizing some of the algorithms proposed to enhance the quality of solutions and improve on their running time as well. We also contemplate using a constraint-programming approach that would, in our view, best fit the constrained character of this type of problem. Finally, we consider implementing a metaheuristic cooperation under a Mozart-Oz environment (<http://www.mozart-oz.org/>) to deal with our problem.

References

1. Abramson, D.: A parallel genetic algorithm for solving the school time tabling problem. 15 Australian Computer Science Conference, Hobert, Feb. 1992.
2. Abramson, D., Dang, H.: School Timetables: A Case Study in Simulated Annealing. Applied Simulated Annealing, Lecture Notes in Economics and Mathematics Systems, Springer-Verlag, Ed. V. Vidal , Chapter 5, 103 - 124, 1993.
3. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaptation of heuristic orderings. Annals of Operations Research, 129, 107-134, 2004.
4. Burke, E.K., De Werra, D., Kingston, J.: Applications in timetabling. In: Yellen J., Gross J.L. (Eds): Handbook of Graph Theory, Chapman Hall, CRC Press., 445-474, 2003.
5. Geraldo Ribeiro Filho, Luiz Antonio Nogueira Lorena, A Constructive Evolutionary Approach to School Timetabling. Applications of Evolutionary Computing, EvoWorkshops2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM. Proceedings, volume2037, Springer-Verlag, Egbert J. W. Boers and Stefano Cagnoni and Jens Gottlieb and Emma Hart and Pier Luca Lanzi and Gunther Raidl and Robert E. Smith and Harald Tijink (editors), 130-139, 2001.
6. Glover, F., Taillard, E., De Werra, D.: A user's guide to Tabu search. Annals of Operation Research, Volume 41, 3-28, 1993.
7. Glover F., Laguna M., Tabu Search, Kluwer, Boston, MA 1998.
8. Merlot, L.T.G., Boland, N, Hughes, B.D.: A Hybrid Algorithm for the Examination Timetabling Problem. E. Burke and P. De Causmaecker (Eds.): PATAT 2002, LNCS 2740, 207–231, 2003.
9. Kragelund, L.V.: Solving a timetabling problem using hybrid genetic algorithms. Software Practice and Experience, 27(10): 1121-1134, Oct 1997.
10. Petrovic, S., Burke, E.K.: University Timetabling. Ch. 45 in the Handbook of Scheduling: Algorithms, Models, and Performance Analysis (ed. J. Leung), Chapman and Hall, CRC Press, 2004.
11. Reeves, C.R.: Modern heuristic techniques for combinatorial problems. Black Scientific Publication, 1993.
12. Ross, P., Corne, D., Fang, H.: Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. In: Schwefel, H.P., Davidor, Y., Manner, R. (Eds.): Parallel Problem Solving from Nature (PPSN) III, LNCS, Vol. 866, 560–565, 1994.

Dynamically Configured λ -opt Heuristics for Bus Scheduling

Prapa Rattadilok and Raymond S K Kwan

School of Computing, University of Leeds, UK
`{prapa, rsk}@comp.leeds.ac.uk`

Bus scheduling is a complex combinatorial optimization problem [3], [10]. The operations planning and scheduling process starts off with the designing of a timetable of trips that have to be served by buses. Each trip has a starting time/location and a destination time/location.

Bus scheduling involves assigning a set of trips to a set of buses such that:

1. The sequence of the trips for each bus is time feasible: no trip precedes an earlier trip in the sequence
2. Each trip is served by exactly one bus

There are two types of operational cost involved, these are layover (idle time) and dead running (relocating the bus between locations without passenger, this includes leaving and returning to the depot). The objective is to minimise the operational cost and the number of buses. Given the nature of the problem it is almost always impossible to reduce the layover and dead running cost to zero. The complexity of the problem quickly increases as the number of the depots and the types of vehicle increases.

A solution is said to be λ -optimal (or simply λ -opt) if it is impossible to obtain a better solution by replacing any λ relation instances by any other set of λ relation instances. The λ -opt heuristic [5] is based on this concept of λ -optimality where in each trial, λ instances of the chosen relation (mapping between problem components, e.g. bus trips to bus's timeslots) in the working solution are exchanged. The trial process continues until a move that satisfies the specified acceptance criteria is found. The accepted move is then used to update the working solution. Computational time rapidly increases for increasing values of λ , as a result, the values $\lambda = 2$ and $\lambda = 3$ are the most commonly used. When $\lambda = \text{number of relation instances}$ we have an exhaustive search where every possibility is tried. In many applications $\lambda = 2$ is powerful enough to yield near optimal solutions in a fraction of the time needed for an exhaustive search.

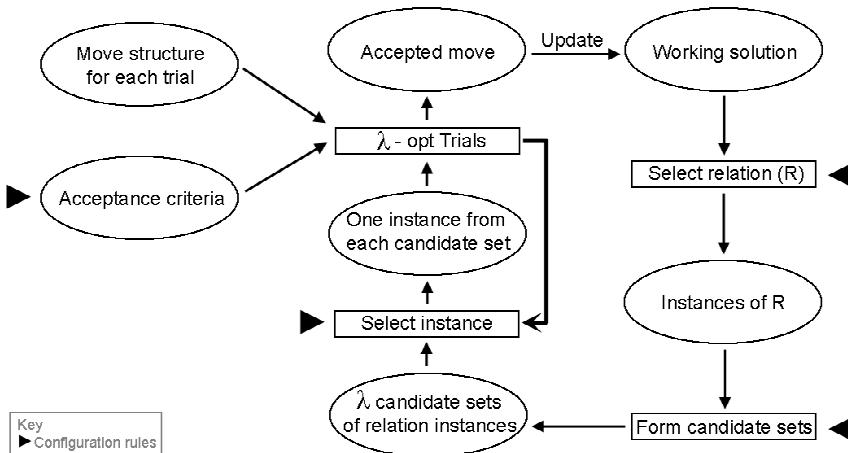


Fig. 1. λ -opt Heuristic

Figure 1 illustrates the general scheme of a λ -opt heuristic. Apart from specifying value of λ , there are four other decisions that need to be taken to fully configure a λ -opt heuristic. The relative merits of the options are not obvious for example, which relation should be selected, which trial solution should be accepted, etc. Different combinations of design options constitute different search strategies. If dynamically configured, these design options could adapt the heuristics to suit the current state of the search process, but choosing “the right” combination of design options is not a trivial task. Figure 2 illustrates the hyper-heuristic framework for the λ -opt heuristic dynamic configuration.

BOOST [4] applies an object-oriented paradigm to solving the bus scheduling problem. Object classes like ‘Vehicle activities’ (represent bus links, the connection between two bus trips) and ‘Link swap rules’ are used. The link swap rules or the scheduling heuristics provide the search moves for the optimization. For example, swapping valid bus links (bus links that correctly follow the sequence) with invalid bus links or valid bus links with start activities links (in order to find earlier starting time). These link swap rules are the extended functions that made up the algorithm in VAMPIRES [8]. These heuristics are the 2-opt scheduling heuristics. Although the major benefit of applying object-oriented models was to increase the extensibility of the application, the heuristics used inside the link swap rules class still require much of the problem specific knowledge. Configuring the 2-opt heuristic use within the system is still a tedious task. The heuristics were hard-wired and the design options were specified by the system designer. A more desirable approach would be to have the design options to be configured by the system itself based on the current search situation.

Hyper-heuristics are a powerful emerging search technology [1]. Search algorithms can be constructed from a collection of simple neighbourhood moves referred to as

low-level heuristics. Rather than hard-wiring such simple moves, hyper-heuristics employ a domain independent driver that iteratively makes dynamic decisions on which simple move or moves should be executed next. The selected heuristics can be knowledge-poor heuristics like simple add, drop and swap moves or complete algorithms more akin to Meta-heuristics.

Soubeiga [9] proposed a choice function based hyper-heuristic driver, which has components designed for search intensification and diversification and incorporates some simple learning capability. The choice function provides the ranking of the low-level heuristics, based on information about the individual performance of each low-level heuristic, joint performance of pairs of heuristics, and the amount of time elapsed since the low-level heuristic was last called. The low-level heuristic performance is calculated in terms of the amount of solution improvement the low-level heuristic has achieved and the time it used to obtain this improvement. This choice function has been applied to select problem specific low-level heuristics on several timetabling and scheduling problems [2], [6].

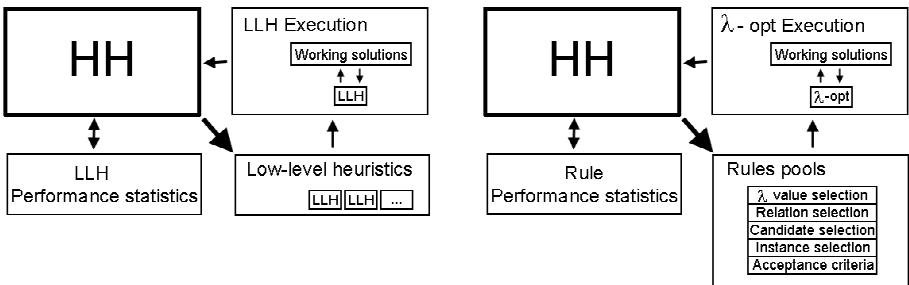
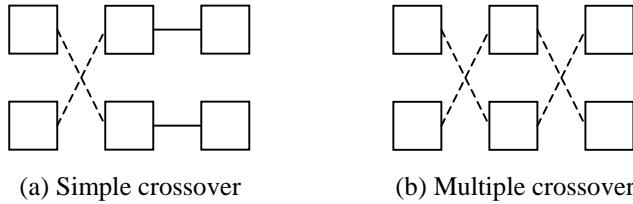


Fig. 2. The Hyper-heuristic framework for the static heuristics (left) and λ -opt heuristic dynamic configuration (right)

There are many possible useful relations for bus scheduling, for example, a relation of multiple bus trips to multiple bus timeslots of the same bus, a relation of a single bus trip to a single bus's timeslot. In the initial experiment, only the simple type of relation is experimented with, a trip to a bus's timeslot. Instead of trying to swap exhaustively between the 2 candidate sets ($\lambda = 2$), the trial swap performs the exchange between a selected candidate instance (the first candidate) and all other candidate instances (the second candidate). The first candidate is obtained by selecting randomly from the top ten instance of the selected ordered candidate set. Three kinds of instance order are used, based on four different kinds of cost (dead run, idle time, start activity and end activity), max-min, min-max and random. After all candidates are tried, the best pair will be selected and the update will be made to the schedule. Figure 3 illustrates the move structures used. Each square represents each bus trip and the dash line represent the bus link where the crossover is made.

**Fig. 3.** Move structures

Similar rules were previously used on a university timetabling problem [7]. A few minor modifications were needed to make it suitable to bus scheduling, for example, the constraint violation in timetabling is considered as the operational cost in bus scheduling. The starting solution is generated using a simple greedy assignment algorithm. The trips are assigned to the bus based on their starting time (which was given in the problem data set). A new bus is started once no more trips can be assigned to the existing bus. The process continues until all buses are tried. If there are any more trips left un-assigned then the rest of the trips are assigned to any empty bus timeslot.

In the initial experiment, all rules are expanded into every possible configuration (every possible low-level heuristic that could be obtained within the provided set of rules). The aim of this initial experiment was to investigate the feasibility of this general framework. At each iteration, the hyper-heuristic selects an expanded combination based on their score (as if it was to select a low-level heuristic).

The initial results obtained on a test data set is comparable to a highly specialised approach [4]. The fact that similar or identical low-level heuristics can be reused demonstrates the main strength of hyper-heuristics, flexibility. Further tuning and optimisation can improve performance but the fact that that isn't needed can be seen as a major success of the system. One possible fine-tuning is to supply hyper-heuristic with a more problem specific rules. More details results will be presented at the conference.

The initial investigation has revealed possible obstacles that need to be overcome before this approach can be considered completely successful. For example, the hyper-heuristic is performing less well as the number of rules increases. Rather than selecting an expanded configuration the hyper-heuristic has to be configured more dynamically. It is believed that this dynamic configuration is too big a task for a single choice function alone. Our current investigation is looking into a hierarchical hyper-heuristic where each layer makes different decisions but all leading to one goal of dynamically selecting the suitable configuration.

References

1. Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., and Schulenburg, S.: Hyper-Heuristics (2003): “An Emerging Direction in Modern Search Technology”. In: Glover, Fred, and Kochenberger, Gary A. (eds.): *Handbook of Meta-Heuristics*. Kluwer Academic Publishers, Boston, USA, 457-474.
2. Cowling, P., Kendall, G., Soubeiga, E. (2002): Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In: Cagnoni, C. et all (eds.): *EvoWorkShops. Lecture Notes in Computer Science*, Vol. 2279. Springer-Verlag, Berlin Heidelberg (2002)
3. Daduna, J. R. and Paixao, J. M. P. (1995): “Vehicle scheduling for public mass transit – An Overview”. In: Daduna, J. R., Branco, I., Paixao, J. M. P. (eds.): Computer-aided transit scheduling, Springer-Verlag 76-90
4. Kwan, R. S. K. and Rahim, M. A. (1999): “Object Oriented Bus Vehicle Scheduling – the BOOST System”. In Wilson N. H. M. (ed.), Computer-aided transit scheduling, Springer, Berlin, pp-177-191, 1999.
5. Lin, S. and Kernighan, B. W. (1973): “An Effective Heuristic Algorithm for the Travelling Salesman Problem”. *Operations Research* Vol. 21, 498-516.
6. Rattadilok, Prapa, Gaw, Andy, Kwan, R. S. K. (2005): “Distributed Choice Function Hyper-heuristics for Timetabling and Scheduling”. In Burke, E. and Trick, M. (eds.): *Practice and Theory of Automated Timetabling V*. Springer LNCS. Vol. 3616/2005 pp. 51-67
7. Rattadilok, P. and Kwan, R. S. K. (2005): “Hyper-heuristic Driven Dynamically Configured λ -opt heuristics”. In: Proceeding of Metaheuristic International Conference 2005
8. Smith, B. M. and Wren, A. (1981): “VAMPIRES and TASC: two successfully applied bus scheduling programs”. In: Wren, A. (ed.): Computer scheduling of public transport. North Holland 97-124
9. Soubeiga, Eric (2003): “Development and Application of Hyperheuristics to Personnel Scheduling”. *Ph.D. Thesis*, University of Nottingham.
10. Wren, A. (1981): “General review of the use of computers in scheduling buses and their crews”. In: Wren, A. (ed.): Computer scheduling of public transport. North Holland, 3-16

A Dispatching Tool for Railway Transportation

Pascal Rebreyend

Computer engineering department, Dalarna University
Röda vägen 3, Borlänge, Sweden prb@du.se <http://www.du.se/~prb>

1 Introduction

Railway transportations are more and more used in many countries for many reasons (environment, costs,...). Like public and freight transportations, reliability (regarding timetables) is a crucial issue. With an increasing traffic, delays occurs more often. To minimize the effects of delays is the main goal of the dispatcher.

2 Problem description

Almost all trains have to run according a schedule. One exception may be some freight trains which may start at a random time. Unfortunately, for some reasons, some minor problems occur and lead to have delays. Due to limited resources available (tracks, cars,...), a small problem can lead to a chain of delays. To monitor the traffic and take decisions are the main goals of the train dispatcher. This person decides which train can run and on which track.

An example is the single-track problem. In this case, the schedule is built in order to let trains going in opposite directions to meet in some specific places (stations, yards,...). If one train is delayed, the dispatcher should decide how to organize the crossing. Basically, in most cases, he has to decide between to keep the crossing at the same location (by delaying the other train) or to move the crossing to another place (but may increase the delay of the already delayed train). The decision is difficult since it may introduce other conflicts or delays later.

Similar problems occur in general networks (dual-tracks,...) where the dispatcher has to choose a compromise. Since improving existing infrastructure is expensive and the traffic is increasing, the work of the dispatcher becomes more and more complex and difficult. Another factor is the wider range of trains used nowadays: suburban trains, freight trains, long-distance passengers, fast trains,... Another complex problem to handle is the connections between trains.

3 Simulation tool

Researchers on this dispatching problem need to have a simulation tool (and a benchmark too) to analyze different methods.

3.1 Extended tool

This tool, called **Distrain**, is an extended tool since its main characteristic is to be open and flexible. This tool is also targeting the general problem, not only single-tracks or specific problems [1]. This openness and flexibility is useful to deal with complex problems and to handle all constraints. An example of constraint is the need to take into account pricing, i.e to charge each train with a price depending of its physical characteristics and which level of priority is used. Therefore, the tool should be able to check for a future train which time will be required, and the potential effect of this train on the future traffic to fix the price.

This tool intends to *help* the dispatcher to choose a solution. Therefore, to be able to propose different possibilities is an interested feature. This can be easily achieved by Genetic Algorithms since in such method there is a population of solutions. Another interesting research issue is to let the software to learn which “kind” of solution is preferred by the dispatcher.

3.2 The model

To achieve this goal, data are represented using XML in different configuration files (networks, timetable, trains,...) [2]. The choice of the model is critical since it is used to represent data.

Data are decomposed into different parts. One part is the *network*. In this part we represent the fixed infrastructure used (tracks, stations, yards, switches,...) with all information needed to be able to simulate the network. The second part of the representation is dedicated to technical specifications of different trains. In order to run a simulation, technical characteristics of the different trains need to be known. By technical characteristics, we mean information like weight, length, power, maximum speed, These informations are needed to compute traveling time (especially time needed to accelerate) as well to know if a train can stop in an uphill section. The third part is the *timetable* where precomputed timetables are stored. Then, we need to represent all specific constraints between trains, or between a train and another mean of transports since inter-modal transports are more and more popular. Dependancies between trains can be called “one-way”, when a train cannot start from a station before the arrival of another one (plus the delay to let passengers to walk), or “two-ways” when two trains needs to exchange passengers and therefore stay together at the station. It’s worth to mention that dependancies can either be with a single or a list of trains (usual case) or with a destination if the frequency of trains on the route is high and therefore it’s worthless to delay such trains. In this second case, the system should only take care at the end of day and maybe delay the last train. The last part needed for the simulation is the list of *unexpected events* which occurs along the day. An unexpected event is composed of three different steps: discovery of a problem, information about the resolution of the event (when the problem will be solved) and resolution of the problem.

3.3 Methods and algorithms

In Sweden, dispatching is done mainly by humans with a computer system to inform them [3]. It is important to notice that they work on a small area. Their work is based on a priority list: on-time time trains have priority over delayed trains, passengers trains over freight trains,... By using a tool working at the country scale, the whole problem can be analyzed which is important with fast long-distance trains. This approach can be implemented using heuristics. But more complex optimization algorithms can be used. Since this tool is based to help the dispatcher to choose a compromise, it's important to have an algorithm which can easily be tuned by changing some parameters and which can provide different solutions. It's why algorithms like branch and bound or genetic algorithms are good candidates. Genetic algorithms are also interesting since they don't need to restart from scratch when a small modification of data is done.

4 Conclusion

The dispatching problem is a complex problem which involved a lot of human factors (passengers) [4]. This problem is connected to other complex problem like infrastructure dimensionning [5], ... At this stage, a prototype is under construction to deal with a section mixing single and dual tracks. This prototype includes simple heuristics.

References

1. Olivera, E.S.d.: Solving Single-Track Railway Scheduling Problem Using Constraint Programming. PhD thesis, School of Computing, The University of Leeds (2001)
2. Rebreyend, P.: Distrain: A simulation tool for train dispatching. In: ITSC, 8th International conference on Intelligent Transportation Systems. (2005) 801–806
3. Hellström, P.: Analysis and evaluation of systems and algorithms for computer-aided train dispatching. Master's thesis, Högskolan Dalarna and Uppsala University (1998) Licentiate thesis, UPTEC 98 008R.
4. Bussieck, M., Winter, T., Zimmermann, U.: Discrete optimization in public rail transport. In Mathematical Programming **79** (1997) 415–444
5. Labouisse, V., Housni, D.: Demiurge: A tool for the optimisation and the capacity assessment for railway infrastructure. World Congress on Railway Research (WCRR), Köln (2001)

Scheduling the Brazilian Soccer Championship

Celso C. Ribeiro¹ and Sebastián Urrutia²

¹ Department of Computer Science, Universidade Federal Fluminense,
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.

celso@inf.puc-rio.br

² Department of Computer Science, Catholic University of Rio de Janeiro,
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil.
useba@inf.puc-rio.br

1 Introduction

The yearly Brazilian Football Championship is the most important sport event in Brazil. It is organized by CBF (the Brazilian Football Confederation) and its major sponsor is TV Globo, the largest media group and television network in Brazil.

The competition is structured as a compact mirrored double round robin (MDRR) tournament [1]. The tournament is played by n teams, where n is an even number. There are $2n - 2$ rounds and each team plays exactly once in each round. Each team faces every other team twice: one at home and the other away. If team a plays against team b in round k , with $k < n$, at home (resp. away), then team a plays against team b away (resp. at home) in round $k + n - 1$.

The organizers and the sponsors search for a schedule optimizing two different objectives. CBF wants to minimize the number of breaks (a team playing two consecutive home or away games, see e.g. [3]) along the tournament (break minimization objective). 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.

2 Problem statement

We consider the 2005 edition of the competition, with $n = 22$ participating teams. Every team has a home city and several cities have more than one team. Some of the teams are considered and handled as *elite teams* due to their number of fans 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 several 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.

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.

Besides following the structure of a MDRR tournament, the schedule should also satisfy other hard constraints:

- Every team playing home (resp. away) in the first round plays away (resp. home) in the last round.
- Every team plays once at home and once away in the two first rounds and in the two last rounds.
- Some pairs of teams with the same home city have complementary patterns, i.e. when one of them plays at home the other plays away and vice versa.
- After any number of rounds, 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).
- Regional games between teams from the same city are not to be played in mid-week rounds or in the last six rounds.
- 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.
- 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.
- Flamengo and Fluminense have complementary patterns in the last four rounds (Flamengo and Fluminense are two elite teams from Rio de Janeiro that share the same stadium for home games).

The two objectives that must be optimized are the minimization of breaks and the maximization of the number of rounds in which there is at least one relevant game to be broadcast to São Paulo plus the number of rounds in which there is at least one relevant game to be broadcast to Rio de Janeiro. Therefore, the broadcast objective regards the number of relevant games that TV Globo is able to broadcast. The break minimization objective establishes the home-away equilibrium in the sequence of games played by each team.

3 Solution strategy and numerical results

To tackle this bi-objective problem, we enforce the break minimization objective to be equal to a tight lower bound and search for a solution optimizing the

broadcast objective. The proof of the tightness of the lower bound is performed experimentally.

The structure of the constraints is such that $4(n - 2)$ is a lower bound to the number of breaks. On the other hand, the broadcast objective is bounded by twice the number of rounds ($4n - 4$) (one game from Rio de Janeiro and another from São Paulo at every round) and by the number of away games of elite teams from Rio de Janeiro against elite teams from other cities plus the number of away games of elite teams from São Paulo against elite teams from other cities.

To solve the problem, we used a strategy similar to the three-phase approach used e.g. in [2]. In fact, we use a four-phase strategy involving complete enumeration, linear programming, and integer programming. We were able to compute an optimal schedule for the 2005 edition of the tournament in less than ten minutes on a standard Pentium IV processor with 256 Mbytes of RAM.

The solution produced by this four-phase approach is better than those produced by the current human scheduler. A quick comparison shows that three constraints were not fully satisfied in the official schedule of the 2005 edition, the number of breaks was equal to 152 and the value of the broadcast objective was equal to 43. In the schedule computed by the four-phase approach proposed in this work, all hard constraints were satisfied, the number of breaks was reduced to 80 and the broadcast objective was also improved to 56 (which is optimal). The software system is able to generate a collection of same cost solutions to be evaluated and compared by the user. The use of the schedules created with the approach proposed in this work is under consideration by the organizers.

References

1. K. Easton, G.L. Nemhauser, and M. Trick. Sports scheduling. In J.T. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 52.1–52.19. CRC Press, 2004.
2. G.L. Nemhauser and M.A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1998.
3. S. Urrutia and C.C. Ribeiro. Maximizing breaks and bounding solutions to the mirrored traveling tournament problem. *Discrete Applied Mathematics*, to appear.

Scheduling school meetings

Franca Rinaldi and Paolo Serafini

University of Udine, Department of Mathematics and Computer Science
`{rinaldi, serafini}@dimi.uniud.it`

We address the following problem: on certain days of the school year parents can meet teachers to discuss about their children. Each parent tries to meet a selected subset of teachers and the meetings are individual. Since in most schools there is no advance planning, parents wait in lines for a long time (one line for each teacher), not only wasting time but also preventing the possibility to meet several teachers.

In this paper we propose a planning method in order to guarantee that each parent meets all required teachers and the wasted time is minimized. A prerequisite for the method to work is that all meetings must last a fixed amount of time, we call *time slot*. Practically, this constraint can be relaxed if a meeting is shorter.

We first consider the problem of minimizing the total time needed for the meetings. This problem can be modeled as a minimum makespan open-shop problem: there is a set I of teachers (machines), a set J of parents (jobs) and each parent j wants to meet a specified subset I_j of teachers (each meeting is an operation). The processing times are $a_{ij} = 1$ if parent j wants to meet teacher i and $a_{ij} = 0$ otherwise.

This particular instance of the open shop problem is polynomial and can be easily solved by means of the algorithm by Gonzalez and Sahni [2] for the open-shop problem with preemption. The minimum makespan can be computed as follows. For each $i \in I$, denote by $b_i := |\{j \in J : a_{ij} = 1\}|$ the number of parents that want to meet teacher i . Similarly, for each $j \in J$, let $c_j := |I_j|$ be the number of meetings required by parent j . It turns out that $\alpha := \max\{\max_{i \in I} b_i, \max_{j \in J} c_j\}$ is the minimum makespan, and the optimal schedule is computed by solving a sequence of α bipartite matching problems, one for each time slot.

Each matching problem assigns parents to teachers. Some teachers and/or some parents may not be assigned in a particular time slot, which, in this case, is called an *idle time* for the teacher and/or the parent. Although the concept of idleness refers in general to both teachers and parents, we consider only idle times of parents as a measure of wasted time. Furthermore, idle times of one particular parent occurring either before or after his/her meetings do not really count as a wasted time. Therefore, from now on we will use the term idle times *only* for the parent idle times in between meetings.

More in detail, the algorithm by Gonzalez and Sahni uses the quantities $\text{slack}(i) = \alpha - b_i$ and $\text{slack}(j) = \alpha - c_j$, which are dynamically updated at each iteration. Elements with zero slack are called *critical* and must be assigned. So

an assignment of parents to teachers is computed under the only requirement that critical teachers and critical parents must be assigned.

The structure of the above algorithm leaves some room to take into account the lexicographically second objective function of the meeting problem, that is minimizing the total number of idle times. The problem of minimizing the number of idle times within a fixed makespan is NP-hard, as can be seen by transforming the no-wait open shop problem $O^g|\text{no-wait}, p_{ij} \in \{0, 1\} | C_{\max}$ [1]. Being the problem NP-hard, we can approach its solution heuristically.

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. Let $W_k(j)$ be the number of idle times assigned to parent j during time slots $\{1, \dots, k-1\}$ by the first $(k-1)$ matching problems. Then each pair (i, j) receives the weight

$$w_{ij} := \begin{cases} 0 & \text{if } s_k(j) = 0 \\ (1 + W_k(j))^2 & \text{if } s_k(j) = 1 \end{cases} \quad (1)$$

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 not been assigned, his/her meetings receive a zero weight. As soon as a meeting for the parent is assigned, the weight rises to one and then increases quadratically with the number of assigned idle times.

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 critic. This introduces inevitable idle times. In order to overcome this behaviour, 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

$$w_{ij} := \begin{cases} -1 & \text{with probability } p_1 \\ 0 & \text{with probability } p_2 \\ 1 & \text{with probability } 1 - p_1 - p_2 \\ c(1 + W_k(j))^2 & \text{if } s_k(j) = 1 \end{cases} \quad (2)$$

where the probabilities p_1 and p_2 and the coefficient c must be properly tuned. A multirun use of this random algorithm has led to remarkable improvements.

However, the solution found this way may have some idle times. In order to improve it we adopt the following local search procedures that modify the schedule of a single teacher and a single parent, respectively. The first one considers only exchanges between meetings of a same teacher. Given teacher i , define the weighted directed graph $G(i)$ having α nodes and, for each $1 \leq t_1, t_2 \leq \alpha$, an arc (t_1, t_2) . This arc is assigned a negative (positive) weight equal to the number

of idle times deleted (introduced) by moving the meeting of i currently scheduled at t_1 to time t_2 . Nodes corresponding to idle times of teacher i are called *sink* nodes. Apparently, if there exists a directed cycle in $G(i)$ of negative cost, the corresponding sequence of switches in the meetings of i decreases the global wasted time by the cost of the cycle. Detecting a negative cycle is easy. If there are no negative cycles in $G(i)$ and there are negative paths ending in a sink node, then we switch the meetings of i according to the corresponding path. The search for such cycles and paths can be repeated, after the necessary updating, until no one exists.

With a symmetric approach, the second procedure tries to eliminate idle times of parent j by performing local exchanges only in the meetings of j . To this aim, define the graph $H(j)$ having α nodes and an arc (t_1, t_2) whenever the teacher i met by j at t_1 is idle at time t_2 . In this case, the meeting (i, j) can be moved from t_1 to t_2 . As a consequence, if there exists a directed path in $H(j)$ from either the first or the last active time of j to one of her/his idle times, the corresponding sequence of exchanges decreases the wasted time of j by 1.

Finally we may also consider to resequence the meetings in order to decrease the number of idle times. This procedure works effectively mainly for solutions with many idle times. We build a complete graph with nodes corresponding to matchings. Any Hamiltonian path corresponds to a sequence of matchings. If we assign to each pair of matchings a cost given by the number of adjacent time slots for the parents, the cost of an Hamiltonian path is equal to the number of all meetings minus the number of all parents minus the number of idle time blocks. An idle time block is a maximal set of adjacent idle times. Since the first two quantities are invariant, a maximum Hamiltonian path provides a matching sequence with the minimum number of idle time blocks. This is however different from minimizing the number of idle times.

References

1. Gonzalez, T. Unit Execution Time Shop Problems *Math. Oper. Res.* **7** (1982), 57–66
2. Gonzalez, T. and S. Sahni Open Shop Scheduling to Minimize Finish Time *Journal of the ACM*, **23** (1976), 665-679

Modelling and Solving the Italian Examination Timetabling Problem using Tabu Search

Andrea Zampieri and Andrea Schaerf

Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica
Università di Udine
via delle Scienze 208, I-33100, Udine, Italy

1 Introduction

The EXAMINATION TIMETABLING problem (ETTP) regards the scheduling for the exams of a set of university courses, avoiding the overlap of exams of courses having common students, and spreading the exams for the students as much as possible.

Carter *et al* [3] provide a set of formulations for ETTP, which differ from each other on some components of the objective function, along with a set of benchmark instances [2]. Formulations and benchmarks by Carter have stimulated a large body of research, so that many researchers (see, e.g., [1,5,4]) have adopted one of the formulations of Carter (or a variant of them), tested their algorithms on the benchmarks, and also added new instances. At present, all instances and the corresponding best results are published on the Web [8].

Unfortunately though, the real-world formulation that applies to our institution and, up to our knowledge, to most Italian universities, differs substantially from Carter's and similar ones. Therefore, in order to solve our practical problem, we have been forced to model and solve the specific situation of our university almost from scratch.

In this paper, we present an ongoing research on the development of a solution software for the ETTP at the school of Engineering of the University of Udine. In details, we present the problem modelling, the structure of the real instances, and the solution algorithm, which is based on Tabu Search [7] and on the interleaving of different neighbourhood relations, in the spirit of [6].

2 Italian Examination Timetabling Problem

An examination session takes place after each teaching term and it is composed by a set of w weeks, and each week is divided in 10 periods of half-day length each (Mon to Fri).

For each session, all courses of the university are split into two sets: courses given in the term preceding the session, called *current courses*, and courses given in a different term, called *past courses*. For current courses there must be *two* examinations in the session, whereas past ones must have just *one* examination.

The examination of each course (either current or past) is of one of the following types:

- Single test: the exam takes place in a single period
- Long test: the exam takes place in the two periods of one single day
- Double test: the exam takes place in two separated tests (typically written and oral), with a predefined minimum distance between them

There are groups of courses, called *curricula*, that have students in common. Exams of courses in the same curriculum should be given in different periods and spread as much as possible (as explained below).

There are also groups of courses, called *clusters*, such that their teachers want to have the examinations in the same period. Obviously, courses in the same curriculum cannot be also in the same cluster.

Exam takes place in the *classrooms*, which are split into three categories based on the size: small, medium, and large. Each exam, or cluster of exams, requires for each test a number of rooms for a given type. Classrooms may also have special equipments, so that some exams may take place only in specific classrooms.

Teachers might declare both their unavailability for a (maximum) number of periods and at most three (ordered) *preferred periods*. Exams cannot be scheduled when the teacher is unavailable and should be scheduled in the most preferred periods. If no preference is declared, all available periods are considered at the same level of preference.

We search for the assignment coherent with the rules for current and past exams, and with the exam types, such that the following *hard* constraints are satisfied, and the violations of the following *soft* ones are minimised.

1. **Strong Conflicts (hard):** Exams of *current* courses in the same curriculum must be all scheduled in different periods.
2. **Room Occupancy (hard):** Two distinct exams (not in the same cluster) cannot take place in the same room in the same period.
3. **Room Capacity and Type (hard):** The number of rooms must be equal to the request. The size must be at least equal to the request. If an exam requires a special equipment, it must be assigned only rooms with that equipment.
4. **Clusters (hard):** Exams belonging to the same cluster must be scheduled in the same period. If two exams in the same cluster have different exam type, then only the first test has to be scheduled together.
5. **Availabilities (hard):** An exam cannot be scheduled when the teacher is unavailable.
6. **Weak Conflicts (soft):** Exams of courses in the same curriculum must be all scheduled in different periods.
7. **Day Conflicts (soft):** Exams of *current* courses in the same curriculum must be all scheduled in different days.
8. **Same-exam distance (soft):** The two exams of the same current course must be scheduled with a given minimum distance.
9. **Distance (soft):** Exams of current courses in the same curriculum must be scheduled in periods with a given minimum distance.
10. **Preferences (soft):** If a teacher has expressed some preferences, exams scheduled in the second, third or not preferred periods are penalised gradually according to a fixed weight pattern.

The objective function is the weighted sum of the soft constraints listed above. Weight are adjusted according to experience and discussions with the dean of the school.

3 Tabu Search for Examination Timetabling

We make use of a tabu search (TS) algorithm. In order to apply TS to our problem we have to define several features. We first illustrate the search space and the procedure for computing the initial state. Then, we define the neighbourhood structure, and finally we describe the guiding search procedure.

Search Space: For each single exam we define one variable that takes the value of the period the exam takes place (or its first test). Constraint types 4 and 5 are always satisfied, the others are included in the cost function (with a high weight) of the search procedure.

Room-related hard constraints (types 2 and 3) are checked and evaluated in each state, but the actual rooms are assigned only in a post-processing step.

Initial Solution Selection: The initial solution is selected assigning each exam at the most preferred period. If there are no preferences, the exam is assigned at random, with constraints types 4 and 5 satisfied.

Neighbourhood Relation: We consider two neighbourhood relations. The first one, called *change*, assigns a new period to an exam (or cluster). The second one, called *swap*, exchanges the periods assigned to two exams (or cluster). Moves that violates constraints 5 are not considered.

Search Techniques: Our algorithm interleaves different *runners* (as defined in [6]): Runners are invoked sequentially and each one starts from the best state obtained from the previous one. The overall process stops when a full round of all of them does not find any improvement. Each single runner stops when it does not improve the current best solution for a given number of iterations (called *max idle iterations*).

We experimented with various solvers that differ from each other on the number and type of runners they use (see below). The base runners are two tabu search using the two neighbourhoods proposed above and a hill climbing using the change neighbourhood; we name them TS(ch), TS(sw) and HC(ch) respectively.

4 Experimental Results

The available dataset is composed by three real-world instances coming from past sessions in our university. The main features of the instances are summarised in Table 1. In details, we show the total number of exams to be scheduled (considering clusters as single courses), the number of periods and rooms, and

the percentage of conflicting courses (all types of conflicts), and the percentage of request of rooms (with respect to rooms \times periods). All data will be made available to the community for comparison through a web page shortly.

Instance	Total Exams	Periods	Rooms	% Total Conflicts	% Room Occupation
1	293	40	17	8.73%	40.85%
2	434	40	17	12.32%	69.29%
3	332	40	18	8.41%	47.76%

Table 1. Features of the Instances

Table 2 shows a comparison of the results of the most promising solvers. For each solver, we run 10 trials for each instance and we report the minimum and the average cost. All trials have been granted 1 minute of computing time.

Solver	Instance 1		Instance 2		Instance 3	
	best	avg	best	avg	best	avg
1 TS(ch)	3098	3350.0	2806	2873.0	3794	4523.4
2 TS(ch) + TS(sw)	2927	3073.2	2573	2596.8	3737	4340.6
3 HC(ch) + TS(ch)	3225	3481.8	2624	2746.0	4166	4658.8
4 HC(ch) + TS(ch) + TS(sw)	2911	3180.6	2502	2569.2	3858	4201.4

Table 2. Comparison of solvers

Table 2 shows that the best results are obtained by solvers 2 and 4, which have the composition of the two TS (without and with HC, respectively). The difference between these two is small, in favour of solver 4.

We conclude showing the results obtained using the *restricted* search space, which is composed by allowing for each exam only the three periods preferred by the teachers (all periods if no preference was expressed). This is the search space underlying the manual construction of the solution used by the university before the development of our software.

Instance	Best	Avg
1	9963	11867.0
2	16045	16613.0
3	19069	20010.6

Table 3. Results using the restricted search space

Table 3 shows the best results obtained using such a search space. It is easy to see that the results are much worse than those presented in Table 2. The fact that there would be a loss of quality was obvious, but the numerical difference

(almost one order of magnitude) highlighted in Table 3 are somewhat surprising. This result shows that the use of the larger search space is necessary, but this was beyond the ability of the human scheduler.

References

1. E. Burke and J. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.
2. M. W. Carter. Carter’s test data. URL: <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/>, 2005. Viewed: June 1, 2006, Updated: June 7, 2005.
3. M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 74:373–383, 1996.
4. S. Casey and J. Thompson. Grasping the examination scheduling problem. In Edmund Burke and Patrick De Causmaecker, editors, *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002), selected papers*, volume 2740 of *Lecture Notes in Computer Science*, pages 232–244, Berlin-Heidelberg, 2003. Springer-Verlag.
5. Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In E. Burke and W. Erben, editors, *Proc. of the 3rd Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2000), selected papers*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer-Verlag, Berlin-Heidelberg, 2001.
6. Luca Di Gaspero and Andrea Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms*, 5(1):65–89, 2006. DOI: 10.1007/s10852-005-9032-z.
7. Fred Glover and Manuel Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.
8. Liam Merlot. Public exam timetabling data sets. URL: <http://www.or.ms.unimelb.edu.au/timetabling>, 2005. Viewed: June 1, 2006, Updated: October 13, 2003.

Optimality aspects with assigning of Magistrates to Sessions and Teams of the Amsterdam Criminal Court

Jan Schreuder¹

¹ dr. Jan A.M. Schreuder, University of Twente, EEMCS-MS, Ra H 207
Postbus 217, 7500 AE Enschede, Netherlands
j.a.m.schreuder@utwente.nl

1 Overview

In the criminal court (Arrondissementen rechtbank, sector strafrecht) of Amsterdam the assignment of magistrates (judges, officers, etc) to sessions needed to handle the cases presented, has become a problem last years mainly caused by the increase of so called mega-sessions. One complicating factor is that there are specialisations amongst the magistrates for sessions at different levels. Another one is that for some (severe) cases a team of three magistrates or judges are required (MK). The assignment takes a period of 4 weeks at a time in which each week up to 100 magistrates and 150 sessions have to be scheduled.

The objective of this research is to develop an optimal decision support system for personnel [1] to work in teams with different functions, organised in different groups. With such a system, a scheduler could make assignments in a shorter time period, more reliable and at least with the same quality. In order to reach for an optimal mix of support and user friendliness against minimal construction time/costs, we used EXCEL (with Visual Basic) for the administration-input-report data representation orientated parts and FORTRAN for the combinatorial assignment parts. CPLEX was used to obtain optimal solutions.

In general the problem described here can be characterized as a problem with a multiple conflicting objective function under overdetermined requirements with both qualitative and quantitative data [2].

The overall optimal assignment approach followed in this applied research is based on three main steps after the input of the relevant data, which is quite a problem in itself. Firstly, a so-called Availability Matrix is developed, which indicates which personnel can be assigned to which tasks, on an individual basis. In former presentations at PATAT conferences [3] the whole administrative system was explained in order to arrive at relevant, robust and reliable data. Database management is crucial here. Pre-processing and reduction rules were applied which reduced the solution space considerably without deleting possible assignments. Next, in order to take into account the team assignments and the working conditions [4], a Combination Matrix is constructed indicating which tasks in the week the personnel can be assigned to. This assignment is still individually based. Finally, the Overall Schedule for the teams is constructed, giving a minimal difference between

the total available working hours and the assigned ones: the objective function. The approach is demonstrated with a (small!) example in Tables 1..4, see Appendix.

The general approach used is to generate a number of possible alternatives and pick the optimal one. In order to arrive always at a solution within a restricted time period we used a crash approach, a Greedy algorithm, a version of the Marriage Problem and an heuristic based on the Branch-And-Bound principle with integer linear programming [5]. The paradox here is that approaches easy to apply give in general solutions far from the optimal one. The more complex the approach the better the solution possibilities will be against a more time consuming character. Another question which is dealt with is the use of commercial available Integer-Linear-Programming packages in parts of the approach.

The optimization part is still in a development stage, but a lot of experiments were executed and the results are promising. The administration part is already in use for some years.

2 Statements

The problem could be characterized using as keywords assignment, timetabling, personnel, magistrates and Decision Support Systems.

The mathematical model is a multiple-conflicting-subjective objective function under overdetermined requirements with both qualitative and quantitative data.

Always it is basic to separate data – model – solvers.

The paradox: the more constraints are added, the faster a (better) solution can be found, however, the bigger the chance on infeasibilities.

You can built in the rules, but you have to check the exceptions.

Better restrict/reduce the solution space than look for better search techniques: use the knowledge on the problem structure. Crash solution?

Collecting robust, reliable and relevant data is hard: 'You get the data you structure'.

You should not solve those problems which you can solve in view of theoretical limitations, but adept to tackle the real world requirements.

References

1. Alves, Maia João and Clímaco, João: A note on a decision support system for multiobjective integer and mixed-integer programming problems. European Journal Of Operational Research, Vol 155, Iss 1 (2004) 258-265.
2. Erol, Ismail and Ferrell, William G., Jr.: A methodology for selection problems with multiple, conflicting objectives and both qualitative and quantitative criteria. International Journal Of Production Economics, Vol 86, Iss 3 (2003) 187-199.
3. Schreuder, Jan: A Decision Support System for Assigning of Personnel to Teams. In: Burke, E.K. an Trick, Michael (eds): PATAT2004, Proceedings of The 5th International Conference on the Practice and Theory of Automated Timetabling (2004) 577-580.
4. Toroslu, Ismail H.: Personnel assignment problem with hierarchical ordering constraints. Computers & Industrial Engineering, Vol 45, Iss 3 (2003) 493-510.

5. Herz, Alain and Widmer, Marino: Guidelines for the use of meta-heuristics in combinatorial optimization. European Journal Of Operational Research, Vol 151, Iss 2 (2003) 247-252.

Appendix: Overall optimal assignment approach

Table 1. Step 1. Availability Matrix: Possible assignments (preprocessing/reduction rules), 0: no assignment possible, 1: possible assignment, 2: specialization.

		Judges												
		AAA	BBB	CCC	DDD	EEE	FFF	PV1						
Sessions	1	2	2	1	2	1	5	Group						
	1.7	3.5	5.0	4.5	5.0	5.0	0		Pt (C)	24.9				
	0	0	0	0	0	0	2.5		Pt (E)	2.5				
Name	F	nr	MK	Pt	Gr	0.7	2.5	4.0	4.5	4.0	5.0	0	Pt (F)	20.7
madMK4A OR	1	2	2.5			0	2	0	0	1	0	0		
madMK4A JR	2	1	2.5			0	0	2	0	1	0	0		
maoSR1	3		1.3			1	1	1	1	1	0	0		
mamPR1	4		1.6			0	1	1	1	1	1	0		
wodMK7A VZ	5	6.7	3.3			0	0	0	2	0	0	0		
wodMK7A OR	6	5.7	2.5			0	0	0	0	0	2	0		
wodMK7A JR	7	5.6	2.5	1		0	0	0	0	0	0	1		
wooSR1	8		1.3			1	0	1	1	0	1	0		
wooPR1	9		1.6			1	0	1	1	0	1	0		
womPR2	10		1.6			1	0	1	1	0	1	0		
dooPREC	11		1.4	2		2	0	0	0	0	0	0		
domPR1	12		1.6			0	1	0	0	1	0	0		
vrmPR1	13		1.6			1	1	1	1	1	1	0		
			Σ	25.3										
						0.4	1.5	3.0	3.5	3.0	4.0		Pt min	
						1.7	3.5	5.0	5.5	5.0	6.0		Ptmax	

Table 2. Step 2. Combinations per person. Need to know about restrictions in assigning points and distance sessions: absmin, absmax, relmax, relmin, ddMK, ddEK (=0.4 6.0 1.0 1.0 2 1).

Fortran output

```

8 UITVOEREN ALTERNATIEVEN
AAA 0.70 1 1.40 dooPREC VZ +
BBB 2.50 1 2.50 madMK4A OR +
CCC 4.00 3 3.80 madMK4A JR + wooSR1 VZ +
        4.10 madMK4A JR + wooPR1 VZ +
        4.10 madMK4A JR + womPR2 VZ +
DDD 4.50 3 4.60 wodMK7A VZ + maoSR1 VZ +
        4.90 wodMK7A VZ + mamPR1 VZ +
        4.90 wodMK7A VZ + vrmPR1 VZ +
EEE 4.00 3 4.10 madMK4A OR + domPR1 VZ +
        4.10 madMK4A OR + vrmPR1 VZ +
        4.10 madMK4A JR + domPR1 VZ +
FFF 5.00 3 4.10 wodMK7A OR + mamPR1 VZ +
        4.10 wodMK7A OR + vrmPR1 VZ +
        5.70 wodMK7A OR + mamPR1 VZ + vrmPR1 VZ +
PV1 0.00 1 2.50 wodMK7A JR +
XXX 0.00 0 0.00 +

```

Table 3. Combination matrix: possible assignment for magistrates. Pt: points available against points required

		judges	AAA	BBB	CCC	DDD	EEE	FFF	PV1
sessions		Pt	0.7	2.5	4.0	4.5	4.0	5.0	0
madMK4A OR	1	2	2.5		2			1	
madMK4A JR	2	1	2.5		2	2	2		
maoSR1	3		1.3			1			1
mamPR1	4		1.6				1		1
wodMK7A VZ	5	6.7	3.3			2	2	2	
wodMK7A OR	6	5.7	2.5					2	2
wodMK7A JR	7	5.6	2.5						1
wooSR1	8		1.3		1				
wooPR1	9		1.6		1				
womPR2	10		1.6			1			
dooPREC	11		1.4	2					
domPR1	12		1.6				1	1	
vrmPR1	13		1.6				1	1	1
			0.7	2.5	3.8	4.1	4.1	4.6	4.9
					4.9	4.9	4.1	4.1	4.1
						4.1	4.1	4.1	5.7
									0

Table 4. Step 3 Construction Overall Schedule

Crash: only special: first still available

Greedy: use combinations, see Table 2

Marriage Problem: ordering assignments, see Table 1, forbidden combinations are needed

Optimisation: as much as possible combinations (ILP)

Always assign just one of the alternative combinations or a part of one combination.

				Crash	Greedy	Marriage	Optimize
				2		1	
madMK4A OR	1	2	2.5	BBB	BBB	BBB	BBB
madMK4A JR	2	1	2.5	CCC	CCC	CCC	CCC
maoSR1	3		1.3		DDD	DDD	EEE
mamPR1	4		1.6		FFF	FFF	FFF
wodMK7A VZ	5	6.7	3.3	DDD	DDD	DDD	DDD
wodMK7A OR	6	5.7	2.5	FFF	FFF	FFF	FFF
wodMK7A JR	7	5.6	2.5	PV1	PV1	PV1	PV1
wooSR1	8		1.3		CCC		EEE
wooPR1	9		1.6			CCC	CCC
womPR2	10		1.6				
dooPREC	11		1.4	AAA	AAA	AAA	AAA
domPR1	12		1.6		EEE	EEE	DDD
vrmPR1	13		1.6		FFF	FFF	FFF
points assigned		$\sum \text{PtG}$	20.7	12.2	19.6	19.9	21.2
% from optimal		ObjF	100	41	5	4	1

Multi-Calendar Appointment Scheduling: Calendar Modeling and Constraint Reasoning

Stephanie Spranger and François Bry

Institute for Informatics, University of Munich, Germany
http://www.pms_ifi.lmu.de/

1 Introduction

This article outlines CaTTS, a language for expressing and solving *multi-calendar appointment scheduling problems* such as planning a phone conference of persons in different time zones and using different calendars. This article complements [1], which is focused on *calendar modeling*, with a presentation of the approach's constraint reasoning.

CaTTS is motivated by the fact that practical scheduling systems should preferably use everyone's calendars, i.e. cultural calendars such as the Gregorian or Hebrew calendars and the plethora of professional calendars, instead of the more abstract temporal specifications commonly used in research.

The 'Calendar and Time Type System' CaTTS consists of a *type definition language* CaTTS-DL and a *constraint language* CaTTS-CL. CaTTS is based on a time model after the set-theoretic 'time granularity' approach [9,10,2,11]. However, in contrast to most time granularity formalisms, CaTTS has no duration-less time points. Instead, CaTTS is purely interval-based reflecting a common-sense notion of time: a time point such as "Tuesday, January 3 2006, 9 a.m.", expressed in the time granularity "hour" ("second", resp.) is internally represented in CaTTS by a time interval with a duration of 1 hour ("second", resp.).

2 Multi-Calendar CSPs in CaTTS

In [2] an approach to point-based metric temporal reasoning with time granularities is described. It allows for modeling and reasoning with simple temporal constraints on points and distances between points in a Disjunctive Linear Relations (DLR) Horn framework.¹ In this framework, one can express constraints like "*at time t (in time granularity g), person A is in London*", but neither "*an event e (in time granularity g) happens during a task t (in time granularity h)*" nor "*an event e (in time granularity g) happens 5 time units (in granularity h) before an event e' (in time granularity k)*". Indeed, expressing temporal constraints like the last two mentioned above require not only time points but also time intervals.

¹ DLR [3] represents temporal constraints by disjunctions of linear inequalities and inequations.

A common approach, followed e.g. by [2], represents a time interval by an ordered pair of time points. Expressing that two such intervals *meet without overlapping* requires to consider both (half or both sides) open and closed intervals. Such intervals, however, are rather counter-intuitive for most users. Furthermore, such a ‘time point-based approach’ makes the modeling of temporal applications significantly more complicated because both, time points and time intervals, have to be considered. Consider an event e taking place sometimes during an interval i which in turn is *during* an interval i' : e also takes place during i' . This example shows that relations on intervals and events must be propagated so as to keep reasoning local. It is much more complicated to maintain locality when time intervals are defined by endpoints than when only time intervals (and no time points) are considered.

The approach reported about in this article avoids the problems mentioned above by considering only time intervals and no duration-less time points. In most practical applications, time points, if needed, are conveniently simulated by small intervals (e.g. in scheduling a meeting, intervals with a duration of 1 second may conveniently simulate time points). This makes it easy to represent

1. finite time intervals using finite domains,
2. quantifications such as “*an event e (in time granularity g) happens 5 time units (in granularity h) before an event e' (in time granularity k)*”, and
3. generalized (i.e. finite and non-convex) as well as (infinite) periodic time intervals

as needed for many applications.

Example 1. A person plans a meeting lasting 3 working days after 20th April 2005 and before May 2005. A colleague’s visit of 1 week must overlap with the planned meeting.

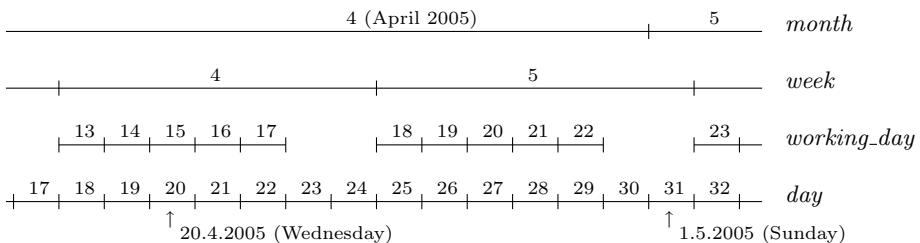


Fig. 1. The calendric types of Example 1.

The *activities* “meeting” and “visit” are represented as finite and convex time intervals that are isomorphic to sets of integers. Activities may refer to different *calendric types* such as “day”, “working day”, “week”, or “working week”. In Example 1, “meeting” refers to “working days”, “visit” to “weeks”, cf. Figure 1.

Using the Description Language CaTTS-DL, the calendar of Example 1 can be expressed as follows:²

```
calendar CalendarExample : Sig =
cal
  type day ;
  type week = aggregate 7 day @ day(-3);
  type month = aggregate
    31 day named January ,
    alternate month(i)
      |(i div 12) mod 4 == 0 && (i div 12) mod 400 != 100 &&
       (i div 12) mod 400 != 200 && (i div 12) mod 400 != 300
      -> 29 day
      | otherwise -> 28 day
    end named February ,
    31 day named March ,
    ...
    31 day named December
    @ day(-90);
  type working_day = select day(i) where
    relative i in week >= 1 && relative i in week <= 5;
end
```

Using the Constraint Language CaTTS-CL, the scheduling problem of Example 1 can be expressed as follows:

```
program SchedulingExample =
prog
  use_calendar unqualified CalendarExample;
  use_format unqualified CatalogExample;
  Meeting is 3 working-day && Visit is 1 week &&
  Meeting after "20.04.2005" && Meeting before "05.2005" &&
  Visit overlaps Meeting
end
```

An *answer* to such a CaTTS-CL program is defined as a consistent Multi-Calendar CSP which cannot be further reduced. The answer to the CaTTS-CL program above is as follows:

```
Meeting is 3 working-day &&
(begin_of Meeting) within ["21.04.2005" .. "22.04.2005"] &&
Visit is 1 week &&
(begin_of Visit) within ["Week04.2005" .. "Week04.2005"]
```

The programmer might ask for one or all *solutions* to this answer. They are computed from the answer by exhaustive search. The solutions to the CaTTS-CL program above are as follows:

```
(Meeting=[ "21.04.2005" .. "25.04.2005" ] && Visit="Week04.2005") ||
(Meeting=[ "22.04.2005" .. "26.04.2005" ] && Visit="Week04.2005")
```

² '@ day(-3)' expresses that week 1 begins with day -3.

3 Solving of Multi-Calendar CSPs expressed in CaTTS

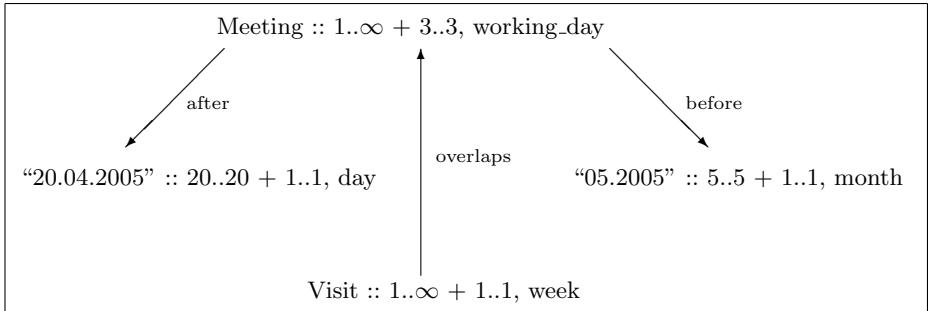


Fig. 2. The scheduling problem of Example 1 as a constraint network.

Multi-Calendar CSPs are seen as *constraint networks*, cf. Figure 2. Since CaTTS makes it possible to use different types expressing temporal granularities such as “days”, “weeks” and “months”, conversions are needed. CaTTS conversions are formalized by a coercion semantics for subtyping. They are expressed by *calendar conversion constraints*. These conversions follow CaTTS’ novel subtyping relations, i.e. *aggregation* (e.g. a week is an aggregation of days) and *selection* (e.g. working days are a selection of days).

In Example 1, the temporal constraint “overlaps” on the variables “Visit” (of type “week”) and “Meeting” (of type “working day”), require type conversions, e.g. to the closest type in the subtyping hierarchy, i.e. “day”. Indeed, a week is a set of 7 consecutive days and working days are selected among all days.

Conversion constraint defines equivalences between calendar domains and make it possible to solve Multi-Calendar CSPs by reducing them to CSPs over finite domains. For example:

$$X :: 1..8 + 7..7, \text{day} \simeq Y :: 1..2 + 1..1, \text{week}$$

The time interval $[8, 14]$ of days represented by X corresponds to the time interval $[2, 2]$ of weeks represented by Y .

CaTTS’ solver for Multi-Calendar CSPs is *complete* in the sense that if it terminates returning a consistent problem, then the original problem is (globally) consistent. Testing for (global) consistency of a Multi-Calendar CSPs is *linear* in both the number of constraints and the number of variables with respect to the size of the calendar domains. This follows from the facts that

1. testing consistency of classical CSPs over finite domain variables where the domains are represented by intervals is linear [8,6], and

2. the access to a CaTTS-DL conversion function in the conversion constraint can be done in *constant time*.

The search for one or all solutions to a bound consistent Multi-Calendar CSPs is however NP-hard.

4 Conclusion

The language CaTTS describes in this article makes it possible to express temporal constraints referring to different calendars. Temporal constraints referring to different calendars are needed in practice.

Novel aspects of the approach are as follows:

1. Only time intervals but no (duration-less) time points are considered,
2. two subtype relations, *aggregation* and *selection*, and
3. the conversion constraints reducing Multi-Calendar CSPs to CSPs over finite domains.

The approach provides with a solution to the problem of “time granularity conversion” mentioned in [12].

Further investigations of the issue would be desirable. Indeed, the approach described in this article cannot solve complex optimization problems like the scheduling of working shifts that require

1. constraints over infinite periodic time intervals and
2. soft constraints [15] and/or preferences [16].

While periodic time intervals can be specified in CaTTS, CaTTS’ solver cannot process them. Soft constraints or preferences cannot yet be expressed in CaTTS.

Acknowledgments

This research has been funded by the German Foundation for Research (DFG) within the PhD Program ‘Logic in Computer Science’ (GKLI) as well as the European Commission and the Swiss Federal Office for Education and Science within project REWERSE (cf. <http://rewerse.net>). The authors thank the anonymous reviewers, Frank André Rieß, and Arnaud Lallouet for useful suggestions.

References

1. Bry, F., Rieß, F.A., Spranger, S.: CaTTS: Calendar Types and Constraints for Web Applications. In: Proc. 14th Int. World Wide Web Conference, Japan. (2005)
2. Bettini, C., Jajodia, S., Wang, S.: Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer-Verlag (2000)
3. Jonsson, P., Bäckström, C.: A unifying approach to temporal constraint reasoning. Artificial Intelligence **102** (1998) 143–155

4. Spranger, S.: Calendars as Types – Data Modeling, Constraint Reasoning, and Type Checking with Calendars. PhD Thesis. Herbert Utz Verlag, München (2006)
5. Allen, J.: Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* **26** (1983) 832–843
6. Frühwirth, T., Abdennadher, S.: Essentials of Constraint Programming. Cognitive Technologies. Springer-Verlag (2003)
7. van Hentenryck, P., Saraswat, V., Deville, Y.: Constraint Processing in cc(FD). Technical Report, unpublished Manuscript (1992)
8. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)
9. Montanari, A.: Metric and Layered Temporal Logics for Time Granularity. ILLC Dissertation Series 1996-02, University of Amsterdam (1996)
10. Jensen, C., (eds.), C.D.: The consensus glossary of temporal database concepts - February 1998 version. (1998)
11. Euzenat, J.: Granularity in Relational Formalisms with Applications to Time and Space Representation. *Computational Intelligence* **17** (2001) 703–737
12. Franceschet, M., Montanari, A.: A Combined Approach to Temporal Logics for Time Granularity. In: Workshop on Methods for Modalities. (2001)
13. Vilain, M.: A System for Reasoning about Time. In: Proceedings of the 2nd National (US) Conference on Artificial Intelligence, AAAI Press (1982) 197–201
14. Meiri, I.: Combining Qualitative and Quantitative Constraints in Temporal Reasoning. *Artificial Intelligence* **87** (1996) 343–385
15. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM* **44** (1997) 201–236
16. Prestwich, S., Rossi, F., K.B.Venable, Walsh, T.: Constrained CP-Nets. In: Italian Conference on Computational Logic. (2004)

How to solve a timetabling problem by negotiation *

Marie-Hélène Verrons¹ and Philippe Mathieu¹

Équipe SMAC, LIFL – CNRS UMR 8022
Cité Scientifique, 59655 VILLENEUVE D'ASCQ, France
`{verrons,mathieu}@lifl.fr`

1 Introduction

Problems involving resource allocation such as appointment taking or timetable creation have usually been tackled as constraint satisfaction problems. For the two aforementioned applications, this forces the users to reveal their agenda in order to give their constraints. But one doesn't always want to reveal his private events. Thus a solution is for the users to keep their agenda private and to use negotiation to take appointments or courses.

In this article, we present a negotiation based approach for timetable creation. Our aim here is not to present an approach that gives the best results but an original one that is more flexible than others which must restart from scratch at each change in the environment. Using agent based negotiation enables us to dynamically add or remove constraints or agents. We show how to transform the timetabling problem into a negotiation problem between agents. Note that an agent can represent a person as well as a thing such as a room.

To illustrate our purpose, we use a university timetable creation application. A benchmark has been proposed by the group Asa GDR-I3 [1]. This problem needs, in order to provide a solution, to be able to adapt itself in response to dynamic changes in the environment. This problem needs a collective search of the solution, and isn't a simulation problem: the aim isn't to recreate virtually the behaviour of an existing organism, but to furnish an expertise.

In order to develop this application, we use a generic negotiation API: GeNCA [2]. This API provides the whole management of negotiation processes and only needs communication configuration and strategy definition. For the application we want to develop, there'll be no work for communication configuration as we'll use the provided MAS communication (either Magique or Madkit). Default strategies are also provided with GeNCA which are well-suited for our application.

In this paper, we first present the negotiation approach for resource allocation problems and the university timetabling creation application that serves our purpose. Then, we present the negotiation toolkit used to build this application, the way to develop it and the results of this approach.

* This work has received a grant from European Community – FEDER – and “Region Nord-Pas de Calais” – CPER TAC –.

2 How negotiation is helpful

2.1 The University timetable creation problem

The problem here is to create the timetables of students and teachers in a University. We present here the benchmark that has been proposed by the group Asa GDR-I3. Actors (in a UML sense) involved are teachers, student groups and rooms. Each one of these actors (individually) has constraints to be satisfied (at best). A teacher has constraints over his availabilities (day of week, time slot), his skills (particular teaching), and his need of particular teaching equipment such as an overhead projector.

A group of students has to follow a particular teaching composed of a set of several courses of several teaching subjects. For example, x courses of subject 1, y courses of subject 2, and so on.

A room is equipped or not of particular equipments (overhead projector, ...) and can be occupied or not during a time slot of a day.

We assume that for each actor, constraints are given in a list. The order in the list gives the importance of the constraint comparing to the others (the first one can be relaxed easier than the last one). The problem to solve consists of conciliating these constraints in order to propose a timetable for a specified duration.

2.2 Which negotiation system are we using?

Many kinds of negotiation systems exist, such as the Contract-Net Protocol (CNP) [3], auctions, multi-step negotiations or else combined negotiations. The negotiation protocol we use here is an extension of the CNP which adds rounds of counter proposals from the contractors and the manager.

The negotiation system we use allows different actors to negotiate contracts over resources. One actor (the initiator) proposes a contract over several resources to a set of actors (the participants). Each participant answers either by accepting the contract or by rejecting it. If the contract has been accepted by a sufficient number of participants, the initiator confirms it. Otherwise, the initiator asks participants which resources they prefer for the contract. He then chooses another set of resources according to participants preferences and proposes a new contract to the participants. This cycle is done until a solution is found or a predefined number of rounds is reached.

2.3 How to transform a scheduling problem in a negotiation problem

The first thing to do is to determine which are the resources to be negotiated and who are the actors in the negotiation. In the application we use here, resources are naturally time slots. Actors are the teachers, the student groups and the rooms.

Then, you have to define the negotiation protocol that best fits your application. For the timetabling problem, the extension of the CNP presented above is the best one. Initiators are the teachers while student groups and rooms are participants. You also specify some features of the negotiation as the possibility to retract yourself from a contract previously taken and to renegotiate this contract. These features are typically needed for timetabling problems.

2.4 Advantages of this approach

The advantages of this approach are twofold. On the one hand, the use of negotiation allows users to find a timetable that respect their constraints without having to reveal them to the others. It also enables them to manage themselves their constraints and so they choose which one to relax if needed. On the other hand, the multi-agent approach facilitates dynamic changes such as the arrival (or removal) of an actor (agent) and negotiation facilitates the changes of constraints. These dynamic changes are taken into account in real time and don't affect the whole process of finding a solution. That is to say that the process has not to be restarted from the beginning but adapts itself to the changes. The resulting system is thus more flexible facing dynamic changes during the resolution process.

3 Coding the problem with GeNCA

To solve this timetable problem, we propose to use negotiating agents in a multi-agent system. We use the MAS platform Madkit and the GeNCA API which provides a framework for building a negotiation application.

The resources that will be negotiated are the time slots. To solve this problem, we decide to assign an agent to each actor. Assume there are 3 teachers and 3 student groups. Thus, 6 agents are defined: $t1, t2, t3, g1, g2$ and $g3$. Agents work on an asynchronous mode. Each teacher inputs his timetable in real time (they use GeNCA as a negotiation help tool), student groups are in an automatic mode, that is to say the agents work as background tasks. Users give priority to resources and to the other actors.

As GeNCA doesn't force to take the contract, it is possible that the whole courses are not scheduled. So with no other features in the agent, if it doesn't succeed in its negotiation, it won't try to move another contract in order to be able to take the one he failed to take. The teacher has to monitor its agent to check that all courses have been taken and otherwise he must cancel courses and move them in order to take the missing ones.

We introduce in the system 3 new agents $r1, r2$ and $r3$ that represent the rooms. Now, teachers have to choose the group and the room when they create their contracts. A teacher can select all rooms when he creates the contract so that he can see if there's a place for his course, choose the room he wants if several are free and search another time slot if no room is free. Selecting all rooms isn't a problem as you can choose to confirm the contract for some of the

participants and cancel it for the others. To do so, initiator strategy might be slightly modified in order to keep only one room participant.

Relaxing constraints and adding new ones are already provided in GeNCA, as you can cancel previous contracts and add new ones. If a contract for a course is cancelled, it will be automatically renegotiated.

Having a multi-agent system enables us to add or remove agent in real time, without perturbing the whole application and having to restart from scratch.

3.1 Concrete results

In our experiments, we have used agents that negotiate contracts that have been created by the user. We haven't added to the agent skills to store a list of courses that must be scheduled and to check that they have been scheduled. As GeNCA doesn't specify that the contract must be taken at the end, it is possible that all courses the user wanted aren't scheduled. The strategy used looks at each possibility of free resources (or resources taken by a less prior initiator) before cancelling the contract but it doesn't cancel another contract in order to take the new one. Each teacher must then check that he has all his courses. To face this problem, we should use an agent having those skills. On the contrary, GeNCA is adapted to dynamic changes: if a course is cancelled, it is automatically renegotiated.

4 Conclusion

In this paper we have presented a novel approach to solve a timetable creation problem. Using negotiation to establish timetables is a new and promising research field. It enables teachers to find a timetable taking into account their constraints without giving them to the others. Moreover, it is a flexible approach that facilitates the integration of dynamic changes such as modification of constraints or arrival of an actor. It is important to keep in mind that this approach is interesting because of its flexibility and not for the results it gives. For the moment, we don't try to give the best results for timetabling problems but we want to show that negotiation is an approach that is worth to be explored. More experiments are needed for the timetable creation problem in order to automate the whole process. We are at the moment designing more specialised agents that will have the list of courses they must schedule in order to make experiments without human intervention for contract creation.

References

1. ASA group members. Emploi du temps. Technical report, groupe ASA du GDR-I3, 2003. <http://www-poleia.lip6.fr/~guessoum/asa.html>.
2. Philippe Mathieu and Marie-Hélène Verrons. A General Negotiation Model using XML. *Artificial Intelligence and Simulation of Behaviour Journal (AISBJ)*, 1(6):523–542, August 2005.
3. R. G. Smith. The Contract Net Protocol : high-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, C-29(12):1104–1113, December 1980.

Experiments with a form of double iterated search for use on hard combinatorial problems with many objectives

Mike B. Wright

Department of Management Science
Lancaster University, LA1 4YX, UK
m.wright@lancaster.ac.uk

1 Introduction

1.1 Iterated Local Search

Iterated Local Search (Lourenço et al., 2003), henceforth to be referred to as ILS, is a metaheuristic that has been used successfully to solve many combinatorial optimisation problems, producing competitive results. The technique essentially consists of two phases: a local improvement phase which leads monotonically to a local optimum and another very short phase which may take one of a number of forms, often involving a small number of moves chosen partly or wholly at random. The technique then iterates between these phases until some stopping criterion is satisfied.

1.2 Problems with many objectives

Problems involving several distinct objectives are often formulated so that there is a single objective function formed by a linear combination of subcosts, each subcost relating to one of the many objectives. However, good metaheuristic methods for the solution of such problems can prove very difficult to achieve, partly because of the often highly complex nature of the solution space. Some metaheuristic techniques aim to address this issue by modifying the weights of the objectives at various times during the search, thus reshaping the new solution space. Such methods include Noising (Charon and Hudry, 1993) and SAWing (Eiben and van Hemert, 1999), which have achieved some successes.

1.3 This paper

This paper reports the results of experiments of a new technique which combines both of the ideas outlined above. The problem addressed is a sports rostering problem.

2 A cricket umpire rostering problem

The problem addressed by this work is that of rostering cricket umpires for the Devon Cricket League in England. This has been solved for several years using a form of Simulated Annealing (SA). However, the problem is a useful test bed for new ideas, since it is sufficiently large and complex (with 14 separate objectives) to be an interesting challenge, yet not so large as to make experimentation excessively time-consuming. A full description of the problem has already been published (Wright, 2006).

3 The double ILS technique

Using intuitive methods for forming an initial solution and defining neighbourhoods, the solution method continues using a form of Double ILS, since there are two loops which use ILS in different ways and for different purposes. It proceeds according to the following pseudo-code.

```

repeat
    set counter = 0
    repeat
        undertake Local Improvement (LI) to a local optimum
        set counter = counter + 1
        if counter < N then make X perturbations
    until counter = N
    change subcost weights
until total number of iterations during LI phases > Z
return to best solution found, reset weights to
original values and carry out final LI

```

The inner loop is a simple form of ILS, while the outer loop is a very different form of ILS (suitable only for problems with many objectives). This is why the technique has been named "Double ILS".

An iteration is defined as the process of calculating the change in cost of a perturbation, whether or not that perturbation is accepted as a result.

The "best" solution means the best using the original weights.

Specifying Z ensures that fair comparisons between experimental runs can be made, since the value of Z effectively indicates the computational effort.

3.1 The Local Improvement phase

The LI technique systematically searches through all possible perturbations, accepting any solution found that has lower cost than the current solution. A device is used to identify perturbations that could not possibly improve the solution; such perturbations are not made, hence reducing the time taken.

3.2 Changing weights

There are two input parameters (H and L). Initially the weights used are the "real" weights and the "control parameter" C is set equal to H . When it is required to change weights, first C is set equal to $H - ((H - L) * I) / Z$, where I is the number of iterations to that point, and then the weights are changed as follows:

Method 1: for each subobjective separately, if its cost is greater than or equal to its value the previous time weights were changed, multiply its weight by R (a random number between 1 and C); otherwise divide its weight by R .

Method 2: decide at random (probability 0.5) whether to multiply or divide the weight by R .

3.3 Experimental results

Experiments to date have $N = 1$ (and thus X is immaterial) and $Z = 500,000$ or $2,000,000$. They show that Method 1 outperforms Method 2 to a small but statistically significant extent; that values of H between 4 and 8, and L between 2 and 4, appear to work best; that the method is already fairly close to being competitive with SA; and that it is considerably better than repeated LI.

Full results will be presented at the conference for various values of N and X , and for different ways of choosing the X perturbations, including totally at random. The overall aim is to find robust values for H , L , N and X which obviate the need for tuning parameters anew for every application.

References

1. Charon, I. and Hudry, O. (1993) The Noising method: a new combinatorial optimization method. *Operations Research Letters* 14(3): 133-137.
2. Eiben, G. and van Hemert, J. (1999) SAW-ing EAs: Adapting the fitness function for solving constrained problems. In Corne, D. et al., editors, *New Ideas in Optimization*, pages 389-402. McGraw-Hill, London.
3. Lourenço, H.R., Martin, O.C. and Stützle, T. (2003) Iterated Local Search. In Glover, F. and Kochenberger, G.A., editors, *Handbook of Metaheuristics*, pages 321-353. Kluwer Academic Publishers, Massachusetts, USA.
4. Wright, M.B. (2006) Case study – problem formulation and solution for a real-world sports scheduling problem. To appear in the *Journal of the Operational Research Society*.

System Demonstrations

Optime: Integrating Research Expertise with Institutional Requirements

Edmund K. Burke^{1,3}, Graham Kendall^{1,3}, Barry McCollum^{2,3}, Paul McMullan^{2,3}, Jim Newall³

¹ School of Computer Science and IT
University of Nottingham
Jubilee Campus, Nottingham NG8 1BB UK
{ekb, gxk}@cs.nott.ac.uk

² School of Computer Science, Queen's University
University Road, Belfast, Northern Ireland BT7 1NN
{b.mccollum, p.p.mcmullan}@qub.ac.uk

³ eventMAP Limited
21 Stranmillis Road, Belfast
j.newall@evmap.com

1 Introduction

EventMAP Limited was formed in 2002 to exploit the commercial potential of scheduling research carried out by the Automated Scheduling, Optimisation and Planning (ASAP) group at the University of Nottingham. The focus of the company is to develop, market and sell examination and course scheduling software into the worldwide Higher and Further Education Sector. We have implemented our systems in Europe, Australia and America. The decision to form a company followed the identification of the obvious market need for a high quality software solution to the scheduling difficulties experienced within the educational sector.

In January 2006 the Company released version 2.5 of the company's flagship examination product, Optime. An earlier version of the software has been presented at an earlier PATAT conference in Konstanz, 2000 [1]. In this paper we discuss the additional functionality made available through version 2.5. The company is in a unique position to integrate leading edge research techniques with the requirements of the user base in the provision of examination timetabling solutions. In the recent international review of Operational Research in the UK (commissioned by the Engineering and Physical Sciences Research Council) [2], a major identified weakness in the current approach to Operational Research is described as follows, "*a gap still remains between the output of a successful research project and what is needed for direct use by industry*" [2]. One of the primary aims of the current efforts by EventMAP Limited is to reduce this gap in relation to examination and course timetabling software. The strategy for achieving this is to highlight the important aspects of the institutional requirements to researchers in the field while continually updating algorithmic techniques within the software, thus enabling solutions to be produced which are both workable and of a high quality.

In general, the aim of improving Optime is to make the system as intelligent and intuitive as possible, providing maximum information to the institutional administrator, allowing him/her to make informed strategic and managerial decisions. The following details the additional functionality available in Optime version 2.5.

2 The Algorithm

The new version of Optime enables the algorithm to be varied depending on the characteristics of the dataset. From observing the relationship between solutions actually used and the characteristics of underlying datasets, it has been concluded that this functionality allows greater institutional control over flexibility within the solution. These observations are the result of a close working relationship with five principal users in the UK and is currently the basis of further research [3]. Currently the combinations of algorithmic structures available are Saturation degree (Heuristic Method) [4], Adaptive [5] and Great Deluge [6] during an additional improvement cycle.

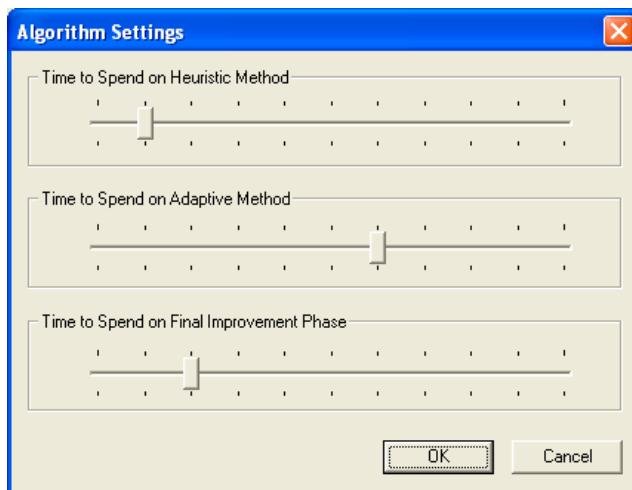


Fig. 1. Construction and Improvement Settings

Figure 1 shows how these algorithms may be varied in the construction of a solution. In essence this means that, in addition to allowing multiple criteria to be set relative to each other during the solution modeling process, the user is now able to adjust or 'direct' the search technique in finding a potential solution.

3 Diagnostic report tool

A report tool has been added to provide information on the current data and solution. This allows each dataset to be analysed for the purposes of provision of information which may be useful in setting up the scheduling model. This is shown under the heading ‘Basic Information’ in Figure 2. This represents a starting point with plans to include in subsequent versions, information on subjects such as projected utilisation of space based on particular chosen formats of the timetabling session. This will help answer common institutional questions such as ‘What is the least amount of time and space that we need to set up a schedule’. Of course, this only serves as an indication as the incorporation of soft constraints adds to the final relevance of the overall solution.

Once a solution is obtained, various items of information are provided as an overview. This is shown under the heading ‘Current Solution Quality’ in Figure 2. More detailed information is provided through the reporting mechanism of the software which will be demonstrated as part of the talk at the conference. An important additional report added in version 2.5 is worthy of note here. This provides the number of students x who have y exams in z days, allowing the user to further understand important student centered characteristics of the solution. An example of the reporting set up interface is shown in Figure 3.

Diagnostics	
Characteristic	Value
Basic Information	
Number of exams	351
Number of students	5045
Conflict density	9.10%
Current solution quality	
Periods used	19
Exams not scheduled	4
Two exams back to back	1650
Two exams in a day	183
OnePeriod	2857
twoPeriods	1342
threePeriods	4875
fourPeriods	4371
fivePeriods	2711

Fig. 2. Dialog Information

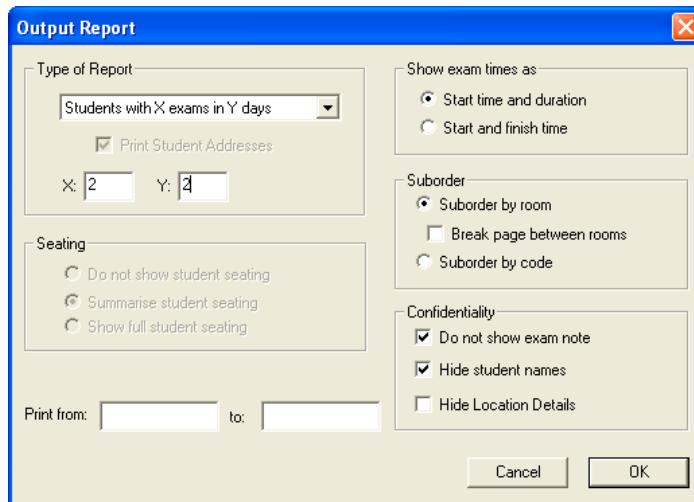
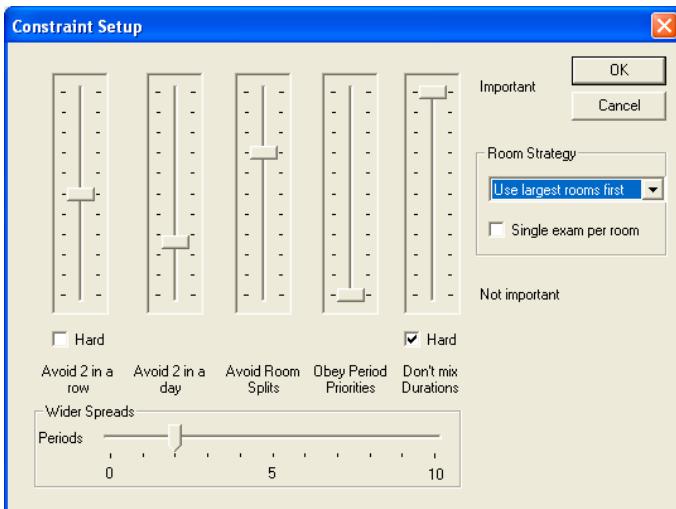


Fig. 3. Reporting

4 Searching for “wider distribution”

Normally, the optimisation function used in the construction of a solution tries to give students the ‘best’ spread or distribution of their exams throughout the examination period. Until now, this has been achieved by a combination of avoiding two exams in a row or in a day. Optime v2.5 allows the user to specify longer periods of time e.g. it attempts to optimise the solution based on examinations being x periods apart, where x is user specified. This will apply a penalty for proximity over the usual two in a day constraint and up to ten periods apart. Nights or days in-between events are not considered solely (as in earlier version) and the closer that two exams are, the higher the penalty will be. Of course, enabling this may degrade other areas of the timetable and consideration to the entire pre-solution modeling must be given serious consideration. The implementation of this functionality can be seen at the bottom of Figure 4. The addition of the wider distribution constraint provides the user with increased control over how a solution is generated. This in effect allows the user to set parameters within the evaluation function in a user centered multi-criteria approach [7].

**Fig. 4.** Optimisation Constraints

5 Special Needs

Students with special needs (e.g. those requiring extra time due to a disability) need to be catered for within the examination timetable. At our leading implementation site, 9% of the students have special needs. These students can now be imported as an extra column on the students and/or enrollments table. These are done via alphabetical codes which must be defined within the system. This is currently used for reporting purposes only though the availability of full functionality for the purpose of timetabling students with special needs (based on a categorization and application of associated constraints) will be available in version 3.0. Those implemented currently are shown in Table 1. These represent further soft constraints which must be considered as part of the provision of a solution.

Special Needs Description	Specified Parameters
Additional Time Required	Minutes
In Separate Room (from)	Room List
Must be Seated in Room	Room List
Avoid Periods	Period List
Students should have no Consecutive Exams	
Students should have a clear Day between Exams	
General Note	Free Text

Tab. 1. Special Needs

6 Future Functionality

The question remaining unanswered in version 2.5 is what happens when a feasible solution is not possible i.e. the specified hard constraints can not be satisfied? Although it is envisaged that the tool should provide a range of reports on how various scenarios would be possible by relaxing hard constraints to soft constraints there is a need for a mechanism of splitting exams into alternatives which may take place at different times. The specification of this functionality is currently at an early stage.

References

1. B. McCollum, J. Newall: "Introducing Optime: Examination timetabling Software", Proceedings of the 3rd international conference on the practice and theory of automated timetabling, pp 485-490, ISBN 3-00-003866-3.
2. EPSRC/ESRC Document Review of Research Status of Operational Research in the UK, 2004.
3. E. K. Burke, B. McCollum, P. McMullan: Examination Timetabling: A new formulation. Submitted to PATAT06.
4. E. K. Burke, J. Newall and R. F. Weare: A Simple Heuristically Guided Search for the *Timetable Problem*, Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (EIS'98), E. Alpaydin, C Fyte (eds), University of La Laguna, Spain 1998, pp 574-579, published by ICSC Academic Press.
5. E. K. Burke and J. P. Newall: Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings, Annals of Operations Research 129, 2004, pp 107-134
6. E. K. Burke, J. P. Newall: Enhancing Timetable Solutions with Local Search Methods. Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling (PATAT 2002), Gent, Belgium, Springer LNCS 2740, pp. 195-206, Aug 21-23, 2002.
7. E. K. Burke, B. McCollum, P. McMullan and J. P. Newall: A preference based measurement of optimization, Internal Technical Report eMAP/2006/02a.

Personnel Scheduling in HARMONY

Laurens Fijn van Draat, Gerhard Post, and Bart Veltman

ORTEC bv, Groningenweg 6K, 2803 PV Gouda, The Netherlands, www.ortec.com,
lfijnvandraat@ortec.nl

The mission of ORTEC, and specifically of HARMONY, is to provide planners and managers access to the benefits of mathematics, and operations research in particular. The basic philosophy of HARMONY is to integrate workforce management with administrative and (other) logistic processes of an organisation and to enrich decision-making within workforce management by OR/MS techniques. The integral approach of workforce management distinguishes HARMONY from most of the studies in the OR/MS literature. Furthermore, HARMONY offers an environment to embed algorithms that add value to planners in the real world.

HARMONY provides an advanced planning environment for shift scheduling and rostering. It offers the user a complete spectrum of support varying from fully manual planning to fully automatic planning. In HARMONY one can for example design cyclic rosters, construct calendar-related rosters based upon these cyclic rosters, make a holiday planning, etcetera.

The planning process starts with the definition of shifts and ends with the realisation of these shifts. In this process one can identify three phases, that are all supported by the mathematical engines in HARMONY. Although none of our clients face the problems of all of these three phases, they all face at least one of them (which is most often the second phase).

The first phase is the definition of the shifts: given a demand for resources per time interval and some constraints for the shifts, one has to construct a minimum set of shifts that satisfies all constraints and fulfils the demand. The engine in HARMONY that supports this phase solves the problem with a set covering algorithm.

The second phase consists of assigning these shifts to the available employees (i.e., create a roster). Here, the user can specify many different types of soft and hard constraints on four different levels (organisation, department, group or individual). For this part of the planning process, HARMONY contains two engines. The first engine uses an insertion technique. It allows for fast construction of good rosters and is ideal for simulation analysis and for completing existing (partial) rosters. The second engine is more powerful and is useful for the construction of complete rosters. It uses a genetic algorithm, local search, and variable neighbourhood search to iteratively improve existing rosters.

In the third phase of the planning process the planner has to specify *where* the employees have to work. The input of this phase is the output of the previous phase, i.e. a roster with the specifications of *when* the employees have to work. The planning in this phase is usually made shortly before it is actually carried out (for example one day or one week). The engine in HARMONY defines this problem as a min cost max flow problem.

In our demonstration, we will briefly go through several aspects of HARMONY, and after that focus on the engines in HARMONY that solve the three problems described above.

References

1. S.L. van de Velde, *ORTEC HARMONY. Innovative advanced planning software for an integrated approach to workforce management optimization*, OR/MS Today, April 2002.

SWOPS (Shift Work Optimized Planning and Scheduling)

Dagan Gilat, Ariel Landau, Amnon Ribak, Yossi Shiloach, Segev Wasserkrug

IBM Haifa Research Lab
Haifa University Campus, Haifa 31905, Israel
shiloach@il.ibm.com

1 The Challenges of Modern Shift Work Scheduling

Assigning workers to shifts is one of the most challenging operational problems in workforce management. The problem is complex, versatile and rich in mathematical, algorithmic and performance challenges. Listed down are some of the major challenges:

1.1 Demand Forecast and Analysis

Demand forecast must take a lot of variables into account – the day of week, the hour of day, the date in the month, special events like holidays and so on. Then there is the analytic part of converting the load volumes into number of agents needed to handle them. The problem get more and more complex when each agent have several skills and the traditional demand graph approach breaks down. Some of the service center, e.g. delivery centers work in a way that is not supported by any clear mathematical model.

1.2 Complex Business Rules

The set of possible business rules is extremely versatile and almost endless. It almost rules out any heuristic approach because of the complex coding effort.

Here is a small sample of rules:

- Pregnant women should not work after 22:00
- Agents that are less than 6 month in the job, should not do night shifts.
- Mr. X should work every day on the 07:30 shift.
- All senior employess should start every day in the same hour, not including weekend days.
- Group X should not do more than two evening shifts per week. (An evening shift is a shift that starts between 14:30 and 20:00).
- Mr. A has a special schedule of 3 hours and 25 minutes, starting at 13:00 every day.
- The night, early morning and weekend shifts should be distributed evenly among all the agents in Group X.
- Nobody should work two nights in a row.
- No more (or no less or exactly) than 10 sales agents should start working between 09:00 and 10:00.

- No more than (or no less than or exactly) 10 agents of group X should be at work at any moment between 20:00 and 22:00.
- The schedule of an agent and his team leader should overlap in at least 80%.
- The 7.5-hour shift should be effective only on Mondays and start only in 08:00 10:00 and 10:30. Only mothers to small children are allowed to do it.
- The first break in the 10 hours shift should start between 120 to 150 minutes from shift start and last 30 minutes. The second break should start between 120 to 140 minutes from the end of the first break.
- Lunch break should be of one hour always between 11:30 and 13:30 and equally spaced among the other breaks.
- There should be at least 2 Russian speakers and 3 Italian speakers between 10:00 and 17:00.
- There should be at least 5% of the marketing agents on-site who speak Spanish at every moment Mon-Fri between 08:00 and 21:00.
- An agent can do 2 shifts in one day only if the first is in the morning (until 08:00) and the second is after 21:00. In any case, he\she should be free for the next day.
- An agent should work between 4 to 5 days a week, with total of at least 38 hours where weekend duties are not included. Weekend limits are open to the user to define.
- If an agent works in weekend they should work in both Saturday and Sunday. They will be compensated in the next week at Thursday and Friday.
- No more than two agents can stay in a break at the same time in Team A, between hours 08:00 and 11:00.
- The sum of all overtime hours per day should not exceed 20, from Monday to Friday, and 10 on weekends and holidays.

1.3 Intra-day Schedule

The traditional schedule requirement was to assign an agent to a shift and tell him what will be their activity during that shift. Now there is a growing request for intra-day scheduling drilled down to one hour resolution. Many centers have a lot of off-line activity that has to be addressed during the shift. Requirements for off-line activity are in terms of total number of hours per week or month. In intra-day planning an agent can do on-line and off-line work in the same shift. Another type of intra-day scheduling is breaks scheduling with the rules that accompany break assignments. Obviously, intra-day scheduling increases the problems size very much and poses performance challenges.

1.4 Workforce Management Analytic Tool

Service centers are not just interested in scheduling. They would also like to regard scheduling tools as decision support systems in more long-term aspects. They would like to be aided in deciding on their shift length and start hours, the combination of skills they need, whether they can save in headcount, how to solve their transportation problems etc.

There is also a growing demand to the ability to run different ‘what if’ scenarios and get some machine generated explanations on questionable results.

1.5 On-Line Crisis Management

Users would like the tool to advise them in case of no-show, sudden rise in demand and other on-line crisis situations. They would like to run a corrective program that solves the problem with minimal changes.

1.6 Complex Working Environment

Large service centers has many independent teams to schedule. They would like the tool to handle several users concurrently and also provide access to the agents themselves to enter their personal preferences. Some of the centers are world wide dispersed and time difference issues must be taken into account. Some of them have huge transportation problems. These problems connect several teams that otherwise can be solved independently.

2 SWOPS as a Workforce Management Solution

2.1 Background

SWOPS is being developed at IBM Haifa Research Lab (HRL) as an on demand service-based system for the shift work optimized planning and scheduling. **SWOPS** is a work force management tool, specifically designed for shift work, supporting multi-service queue, multi-skill environments. It is currently deployed in two very large IBM service centers in Germany and India and going to be deployed in two delivery centers in India as well. **SWOPS** is a follow on to CCS (Call Center Scheduler) which is in production in several directory assistance and banking call centers for more than 10 years.

2.2 SWOPS Technology Aspects

2.2.1 MIP Modeling with Flexibility and Calibration features

The scheduling problem is modeled into a 0-1 assignment problem of agents to scheduling slots. Penalty variables (variables that appear in the objective function) relax the business rules equations and enable the implementation of rule preference and system calibration. In some coverage equations there is also a need to model a non linear behavior of rule violation. It is done by the introduction of several layers of penalty variables to the same equation with increasing penalty values. In fact **SWOPS** uses two calibration mechanisms. The first is controlled by system administrator and

the second is user controlled. The first calibration is among different business rules and the second is between different records of the same business rule.

2.2.2 Business Rules Modeling

Scheduling problems are notorious for their rich diversity and complexity of business rules, varying considerably from one user to another. *SWOPS* offers an extensive ‘library’ of implemented business rules that cover all our current customers’ needs. A lot of experience has been accumulated in modeling business rules and internal modeling tools have been developed. The result is that the implementation of new rules, even the most bizarre ones, becomes quite a simple task.

2.2.3 Flexible Scheduling Environment

A schedule run is characterized by the scheduling time range, a set of active business rules (called ‘*profile*’), group of agents to be scheduled and solver parameters (max. running time, search parameters, node selection etc.). In many cases the full schedule is done by running a sequence of several individual runs, each of them with its unique profile. This is called a ‘*phased run*’ which is a powerful tool for achieving high quality schedules and fast performance.

2.2.4 Advanced Demand Graphs Coverage and Intra-day Scheduling

Usual service queue demand graph coverage assumes that an agent is assigned to a single service queue during the entire shift. *SWOPS* advanced coverage scheme models a business environment in which an agent can handle different service queues at the same shift and even at the same time interval. The new approach to service queue coverage addresses three very interesting problems:

1. The set of skills is large and each agent has a ‘random’ subset of skills (which often occurs in language skills).
2. Each queue has a very small number of incoming calls and requires a small number of agents. If they are devoted to that queue only, they are idle for most of the time.
3. The model enables the breakup of a shift into basic (30 minutes or so) time intervals. An agent may be assigned into several on-line and off-line assignments within the same shift. In fact there is a special assignment variable to each time interval.

This new coverage approach and off-line intra-day scheduling increases the problem size significantly. A considerable reduction in the solving time is achieved by integer variable prioritization that exploits the semantics of the variables.

2.2.5 Delivery Center Modeling

Delivery center deals with off line customer problems rather than phone calls. Handling time is measured in hours or days rather than seconds. The same customer problem can be handled by several people and not continuously and therefore switching times have to be taken into account. There are deadlines and periods of time where the problem is transferred to external bodies and then returns back and so on. The model is too complex for mathematical analysis and therefore there is no form of demand graph to follow. *SWOPS* applies simulation techniques to handle such cases.

2.2.6 Optimized Break Assignment

SWOPS handles timed breaks (that are limited by absolute time end-points such as lunch) as well as ordinary breaks with various spacing rules within each shift. Breaks computations must also take the coverage requirements into account. Since there are quite complicated breaks rules, the implementation uses column generation.

2.2.7 Performance Issues

Performance is a great issue in scheduling. As demands grow, the processing time increases accordingly. Our main accelerators are problem phasing, variable prioritization, and an efficient local improvement phase called ‘cleanup’.

2.2.8 Explainer

The explainer is a tool to answer user questions like: Why such a schedule was computed when they can easily point out a local improvement. The explainer enforces the user proposal and compares it to the computed schedule. If it is cheaper then a bug was found, otherwise (most of the cases) the difference in penalties is displayed to the user in a way they that can immediately recognize. Another use is to find out why a certain business rule has been violated while there seems to be some work around that does violate it.

2.3 Other Tool Features

2.3.1 Demand Forecast and Demand Graph Generation

The demand forecast takes all the day, hour, week month parameters into account + calendar special dates. The demand graph generation uses Erlang C queues model. When there are several activities, a demand graph each generated for each one.

2.3.2 Input of Personal Availability Constraints

Call center agents have their own short and long term availability constraints. They can be inserted via web interface by the agents themselves. The scheduler just has to review and approve or decline them but the input time is saved, (quite significant when you have several hundreds of agents).

2.3.3 Multi-User Environment

Current call centers consist of several independent scheduling units. Each schedule unit (or ‘team’) has its own agents. *SWOPS* is designed as a multi-user system. Just one installation is required for the whole call center. Different teams can work concurrently. The GUI is web based.

2.3.4 Security and Roles

SWOPS provide security facilities according to predefined roles. The typical roles are: Developer, system administrator (one for whole call center), scheduler (one per each schedule unit, in charge of producing weekly or monthly schedules) and agent.

2.4 Future Developments

- Delivery Center Full Solution
- Intra-day Planning - Full solution
- Transportation Solutions.
- Crisis Management.
- Shift Structure Planning

Dialog-Based Intelligent Operation Theatre Scheduler

Karl-Heinz Krempels and Andriy Panchenko*

RWTH Aachen University,
Computer Science Department - Informatik IV,
Ahornstr. 55, D-52074 Aachen, Germany
`{krempels,panchenko}@cs.rwth-aachen.de`

1 Introduction

Medical treatments planning and surgical operations scheduling are substantial elements of hospital management. Operations theatre scheduling deals with assignment of limited hospital resources (rooms, doctors, nurses, etc.) to jobs (patient treatments, surgery, etc.) over the time in order to perform tasks according to their needs and priorities, and to optimize usage of hospital resources [7]. The whole process is restricted by lots of constraints, limitations and preferences [2]. It is characterized by high complexity, which is caused by the uncertainty between the offered capacity and the true demand. As emergency cases occur the planning requirements will change. Existing industrial schedulers usually assume a predefined workflow. Furthermore, they do not take actors' preferences into account and are therefore suffer from levels of non-acceptance of their resulting schedules [2]. State of the art in this domain are the following methods: no planning, pen and paper, non-intelligent tool-based. Therefore, typically scheduling is done manually and involves specialized persons to facilitate the process. An example of an optimization that can be achieved using process automation for nurses rostering¹ is discussed in [4]. In the considered hospital one person spends 3-5 full-time working days on producing a nurses' schedule for the period of one month (only shift assignment). A use of a dialog-based semi-automated system is preferred against fully manual or fully automated systems. This is because of the inability of the later to recognize the changes in a high dynamic environment and to take the responsibility for decisions made. As it has to be possible to add new tasks in the planning process "on the fly" and to adequately plan new situations, we involve a human planner in the scheduling activity. The planner acts as a "sensor" to identify changes as they occur and integrates his knowledge as well as decision-making competence into the planning process.

The proposals for the schedules are made with the help of heuristics. Actual problem solving mechanisms of the system and their evaluation are presented

* This work was supported by the German Research Foundation in the research priority programme SPP 1083 – *Intelligent Agents in Real-World Business Applications*.

¹ rostering and scheduling are used as synonyms here

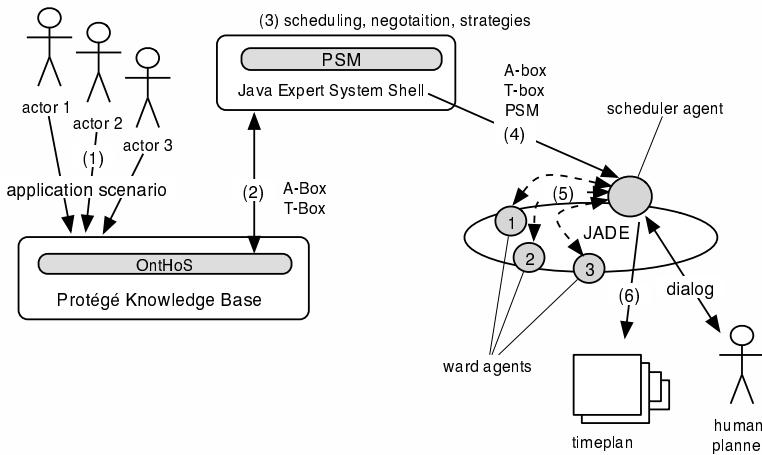


Fig. 1. System Overview

in [5]. Here we shortly describe a scheduling development framework and the constructed prototype for preference-based operation theatre scheduling [6]. Beside that, the paper covers tools used to develop the scheduling system. The framework is integrated in the framework for multi-agent systems *Agent.Hospital* [3].

2 Framework and Prototype Description

We divide the scheduling problem into four sub-problems (shift assignment, team building, job scheduling and room assignment) [6, 5]. For each of the identified subproblems a heuristic was developed that satisfies all constraints and takes into account preferences (like preferred working time, preferred colleague, etc.) of all actors (ward personnel as well as patients). The analyzed application scenario is modeled with Protégé². The outcome is the task ontology [6] (see Figure 1, step 1). Ontology is an explicit specification of conceptualization. It formally defines objects in the scenario and relations among them. Domain ontology OntHoS [1] was used as a reference to develop the own task ontology. The concepts (T-Box) together with its instances (A-Box) were exported into the rule-based expert system JESS³ as facts and rules (step 2). Rule-based expert system is used as an environment where the Problems Solving Methods (PSM) are implemented. These are scheduling heuristics and conflict solving mechanisms (step 3). It is then embedded into the multi-agent system JADE. JADE⁴ is a FIPA-compliant multi agent system (MAS) that is used as a middleware. The outcome is the Jes-

² Protégé Home Page. <http://protege.stanford.edu/>.

³ Java Expert System Shell. <http://herzberg.ca.sandia.gov/jess/>.

⁴ Java Agent Development Environment, <http://jade.cselt.it/>.

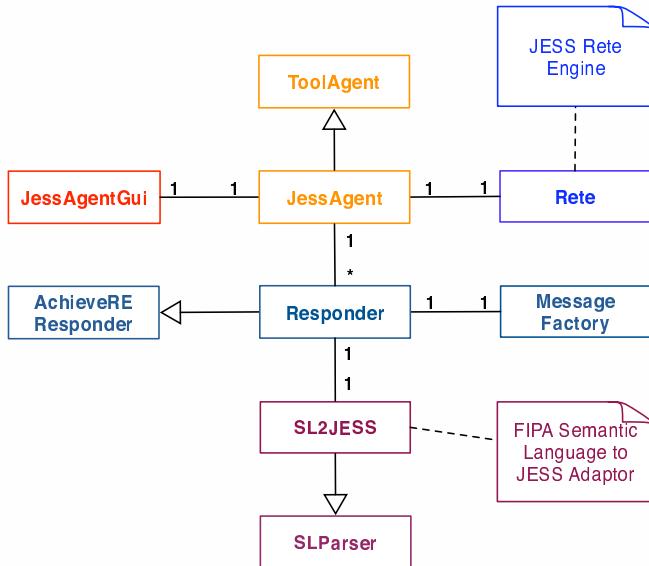


Fig. 2. Diagram of the JessAgent

sAgent⁵ - a JADE-agent that has an embedded Jess engine (step 4). The ability to reason and act rationally is programmed in a rule-based expert system. Communications is facilitated by using a MAS. Further, all the agents (Scheduling Agent and Ward Agents) are started and the scheduling process initiated by the Scheduler Agent. The user interface for interaction with the planner as well as the generated subplans (based on ontological constraints) are provided by this agent. New scheduling tasks are added to the system by a wards' representatives with the help of Ward Agents (step 5).

The deployed Scheduling Agent is based on the JessAgents (see Figure 2), chosen for its suitable integration in the optimized development process. This means that the problem solving methods and the behavior of a JessAgent are written at a higher programming language, which does not require source code compilation. Task ontologies and problem solving methods are loaded at runtime as well as the new fact base, describing a possible change in the considered scenario. Inside of the JessAgent all the received messages (which are in FIPA-SL format⁶) are translated by the SL2JESS adaptor to JESS functions, evaluated in JESS and the answer is automatically generated by the MessageFactory object with respect to the used interaction protocol, speech act, content language and the agent's fact base. All the rules, facts, and functions of the agent can be accessed by the developer through the agents GUI.

⁵ JessAgent Home Page. <http://www-i4.informatik.rwth-aachen.de/agentcities/>.

⁶ <http://www.fipa.org/>.

3 Summary and Outline

This paper gives a short overview of the scheduling framework and tools, used to create a dialog-based semi-automated operation theatre scheduler. Staff timetables in medical departments are subject to lots of constraints, restrictions, and preferences. Scheduling of hospital personnel is particularly challenging because of different staffing needs on different days and shifts, uncertainty between the offered capacity and the true demand, impossibility to predefine treatments' workflow.

We have divided the original problem into four sub-problems and provided a preference-based adaptive heuristics for each of them as described in [6, 5]. The system makes a schedule proposal and it is up to the responsible human actor either to accept, accept parts of the proposition, or to reject the schedule. The prototype was used for the comparison of different heuristic approaches. This allowed to conclude that the proposed algorithms bring a substantial improvement regarding the number of fulfilled wishes of the actors while selecting shift staff and team building, and helps to save expensive human resources that are currently used in hospitals for the manual scheduling. However, the preferences of the involved actors were randomly generated. Evaluations in a real-world setting would be of a great interest and will be made in cooperation with the university hospital.

References

1. M. Becker, C. Heine, R. Herrler, and K.-H. Krempels. OntHoS - an Ontology for Hospital Scenarios. *First International Workshop on Agent Applications in Health Care, Barcelona, Spain*, February 2003.
2. M. Becker, K.-H. Krempels, M. Navarro, and A. Panchenko. Agent Based Scheduling of Operation Theatres. *EU-LAT eHealth Workshop, Cuernavaca, Mexico*, pages 220–227, December 2003.
3. Stefan Kirn, Christian Heine, Rainer Herrler, and Karl-Heinz Krempels. Agent.hospital - agent-based open framework for clinical applications. In *WET-ICE*, pages 36–41, 2003.
4. Lars Kragelund and Torben Kabel. *Employee Timetabling: An Empirical Study of Solving Real Life Multi-Objective Combinatorial Optimization Problems by means of Constraint-Based Heuristic Search Methods*. Master's Thesis in CS, 1998.
5. Karl-Heinz Krempels and Andriy Panchenko. An Approach for Automated Surgery Scheduling. In Edmund Burke and Hana Rudova, editors, *Practice and Theory of Automated Timetabling VI: Sixth International Conference, PATAT 2006 Brno, Czech Republic, August 30 - September 1, 2006, Selected Revised Papers*, LNCS, Heidelberg, 2006. Springer-Verlag GmbH.
6. Andriy Panchenko. *Preference-Based Scheduling of Operation Theaters*. Master's Thesis in Computer Science, Department of Computer Science IV, RWTH Aachen University, Germany, 2005.
7. D.G. Rajpathak. Intelligent Scheduling - A Literature Review. Technical Report KMI-TR-119, Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK, August 2001.

Process Plan Optimization using a Genetic Algorithm

Fabian Märki^{1,2}, Manfred Vogel¹ and Martin Fischer²

¹ Institute 4D Technologies and DataSpaces, Fachhochschule Nordwestschweiz
Klosterzelgstrasse 2, 5210 Windisch, Switzerland
manfred.vogel@fhnw.ch

² Department of Civil and Environmental Engineering
Center for Integrated Facility Engineering, Stanford University
Building 550 Room 553H, Stanford CA 94305-4020, USA
{fischer, markif}@stanford.edu

In this paper we present our ongoing research project GAPO (Genetic Algorithm Process Optimization) that focuses on the development of an optimization module for process plans.

GAPO is part of a 4D-Toolbox that conflates different modules for the 4D planning of construction projects. Among other modules, the 4D-Toolbox consists of a DES (Discrete Event Simulator) that automatically sequences activities into a network plan ³ taking structural and process constraints into account. Thereafter, GAPO is used to optimize the generated network plan in terms of time, cost and resource management forming a multiple criteria optimized process plan ⁴ This process plan can then be joined with the 3D-Model of the construction project - forming a 4D Model - and visualized through the 4D-Player, another module of our 4D-Toolbox.

GAPO is based on a Genetic Algorithm (GA) approach to perform its optimization. GA's are a class of heuristic search methods based on the Darwinian principle of evolution. They mimic and exploit the genetic dynamics underlying natural evolution to search for optimal solutions of general combinatorial optimization problems [1].

Our Evolution Model starts with an initial population of randomly generated process plans. A subsequent population will then be assembled using five strategies which can be weighted by the user. A fraction q of the best individuals will be directly passed to the next population. This guarantees that the quality of the most suited candidates will monotonically increase from generation to generation. A second fraction r of individuals will be passed to the next population after a mutation. On one side, this process opposes early convergence in a local optimum and thereby helps to open new search regions. On the other side, it also allows a fine tuning of suitable solutions by applying small chances on them. A third fraction s of the new population is created by recombining individuals from the old generation. This process forces convergence into an optimum. A fourth fraction t is created by recombining individuals but instead of passing them di-

³ Sequenced activities with predecessor relationships.

⁴ Network plan with time and resource allocations.

rectly into the new population the new individual is mutated beforehand. Last, a fraction u of the new population is created randomly. This process also helps to open new search regions and prevents early convergence in a local optimum [2].

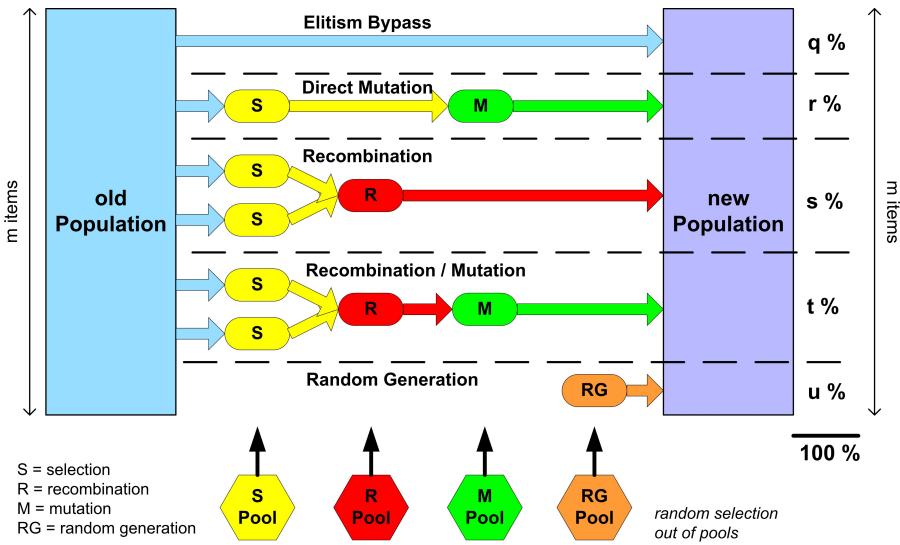


Fig. 1. GAPO Evolution Model

The data structure of a process plan genotype is kept very simple. At the moment, it consists of three arrays. The first array represents the sequence of how the scheduling algorithm should schedule the activities. The second array defines the position where the scheduling algorithm starts to search for a location where the activity can be performed without violating a resource constraint. By definition, this position is between the end of the latest predecessor of the handled activity and the end of the latest scheduled activity. If it is not possible to schedule the activity within this range, it will be added at the end of the schedule as the latest activity. The third array defines the amount of the resources that are assigned to the activity. This representation allows us to use standard mutation and recombination operators [3]. Furthermore, a recombination can also be done by recombining arrays of different individuals.

In the presentation we will show how GAPO is used to optimize construction process plans. This will be done by using our first feasibility study which is based on a real construction project.

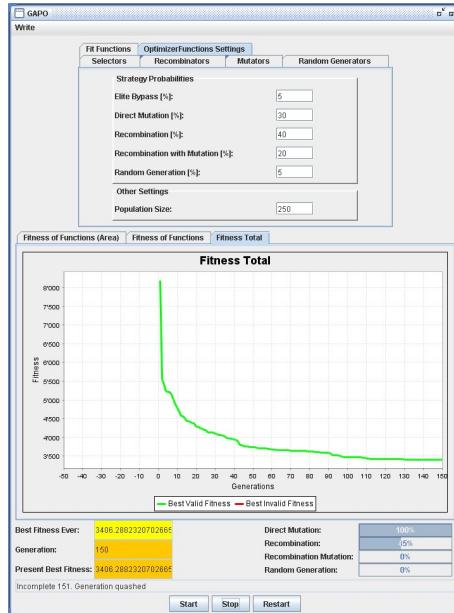


Fig. 2. Screenshot of GAPO's user interface

To further push research on process plan optimization, collaboration between the CIFE⁵ and the *i4Ds*⁶ has been established. One of the goals addressed is the development of a GAPO module allowing the optimization of arbitrary process plans. Furthermore, we will also introduce more mechanisms that enable the user to give the optimization a desired direction and consequently have more control over the outcome of the optimization.

Beside the optimization itself, we also work on the integration of a variation study that will take place during the optimization process. The idea of this variation study is to enable GAPO to take decision possibilities into account and give the user a feedback about which decisions would be favourable in terms of process planning [4]. In the current development cycle, we consider the following variations:

- A change in the quantities of the resources used within a resource group
- A change in the resource group assigned to an activity

Candidates for further variations are:

- A change in the process method used to perform a task

⁵ Center for Integrated Facility Engineering, Stanford University

⁶ Institute 4D-Technologies and Data Spaces, University of Applied Sciences Northwestern Switzerland

- A change in the design of the construction project

Our long term goal is to provide a GAPO module that can be handled by the user as a black box, not requiring any knowledge about optimization, Genetic Algorithms and the like. This would simplify its usage and would also allow using it in other project planning applications.

References

1. Coley, D. A. An Introduction to Genetic Algorithms for Scientists and Engineers. World Scientific Publishing, 1999.
2. Gonçalves, J. F., Mendes, J.J.M., Resende, M. G. C. A hybrid genetic algorithm for the job shop scheduling problem. AT&T Labs Research Technical Report TD-5EAL6J, AT&T Labs Research, 2002.
3. Chevreux, B. Genetische Algorithmen zur Molekölstrukturoptimierung. Universität Heidelberg, 1997. <http://www.chevreux.org/diplom/node5.html> Last retrieved 05/08/2006.
4. Kam, C. Dynamic Decision Breakdown Structure Ontology, Methodology, and Frame-work for Information Management in Support of Decision-Enabling Tasks in the Building Industry. Civil and Environmental Engineering of Stanford University, 2005.

THOR¹: A tool for school timetabling

Fernando Melício^{1,3}, João P. Caldeira^{2,3}, Agostinho Rosa³

¹ ISEL-IPL, R. Conselheiro Emídio Navarro, 1900 Lisboa, Portugal
fmelicio@deea.isel.pt

² EST-IPS, R. Vale de Chaves-Estefanilha, 2810 Setúbal, Portugal

³ LaSEEB-ISR-IST Av. Rovisco Pais, 1, TN 6.21, 1049-100 Lisboa, Portugal
{caldeira, acrosa}@isr.ist.utl.pt

Abstract. This system is the result of our previous work on the subject of school timetabling. It was designed to respond mainly to Portuguese schools from various educational levels. It consists of three main blocks; a graphical user interface; an automatic scheduler and a relational database. This system is now in use by more than 100 schools in Portugal with significant success (<http://www.fmaisMais.pt>).

1 Implementation Framework

School timetabling is a classical combinatorial optimization problem which is associated with a set of constraints. It consists of assigning a set of lessons to time slots within a time period (typically a week), satisfying a set of constraints of various kinds [3]. The constraints that we have used are related to Portuguese schools and are the result of multiple discussions we had previously with people from several Portuguese schools. In Table 1 it can be seen a summary of the constraints we have considered in solving this problem.

The main issue regarding the constraints is that each user can weigh each constraint with a particular value and in this way different schools may have different values to each constraint.

A lesson is the teaching unit. It is characterized by the triple $\langle T^*, C^*, S^* \rangle$. Where T^* is a subset of the teachers set, C^* is a subset of the classes set and S^* is a subset of the subjects set. Each lesson has a duration measured in time slots.

There are two types of lessons:

1. *Simple lesson.* Where $|C^*| = 1$ and $|S^*| = 1$.
2. *Compound lesson.* Where $|C^*| \geq 1$ and/or $|S^*| \geq 1$.

¹ In Portuguese THOR stands for Tabelas Horárias which can be roughly translated by timetabling charts.

<i>Constraint</i>	<i>Description</i>
C_0	Number of time slots of lessons that aren't yet scheduled
$C_{1,2}$	Number of time slots of overlapped lessons (1-classes; 2-teachers)
$C_{3,4}$	Number of time slots exceeding the maximum allowed per day (3-classes; 4-teachers)
$C_{5,6}$	Number of time slots exceeding the maximum consecutive time slots allowed (5-classes; 6-teachers).
$C_{7,8,9}$	Number of preferable time slots filled (7-classes; 8-teachers; 9-subjects).
$C_{10,11}$	Number of idle time slots (10-classes; 11-teachers)
C_{12}	Number of time slots of lessons without a room assigned.
$C_{13,14,15}$	Number of time slots that are forbidden and are filled with lessons (13-classes; 14-teachers; 15-subjects)
C_{16}	Total number of teaching days for teachers
C_{17}	Number of repetitions of lessons of the same subject in the same class per day
C_{18}	Number of time slots that doesn't satisfy the predefined space between lessons.

Table 1. Constraint set

In general, a *compound lesson* means that we have several classes joined together to attend a certain subject or it means that a class can be subdivided into subgroups to attend special subjects, like laboratories, etc.

It is associated with each subject the kind of room it must have, i.e., the resources that there must exist in the room for a lesson of that subject should happen.

This software tool was designed to respond mainly to Portuguese schools from various educational levels. It is based on a modular implementation, and consists of three main blocks; a graphical user interface; an automatic scheduler and a relational database (Fig. 1).

It was developed in C++ using an object oriented technique. It runs on Microsoft Windows® and the database is implemented in Microsoft Access®.

There were two main objectives with the development of this system:

1. It should be fairly easy to interact with it.
2. It would generate good timetables in an automatic way.

The graphical user interface (GUI) consists of several forms where it is possible to enter the school data (i.e., teachers, rooms, classes, subjects), and also change the individual schedules.

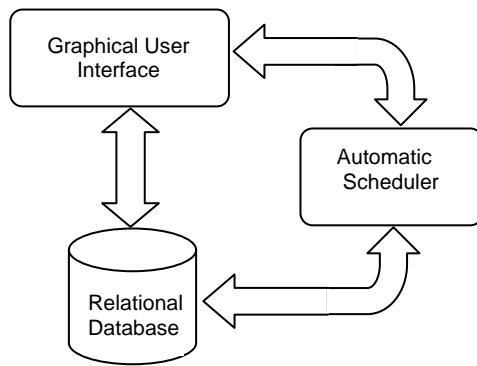


Fig. 1. Block diagram of the implemented system.

An individual schedule looks like an Excel sheet where there is always one or more time slots (cells) selected and we can do all the basic editing operations like delete, insert, undo, etc.

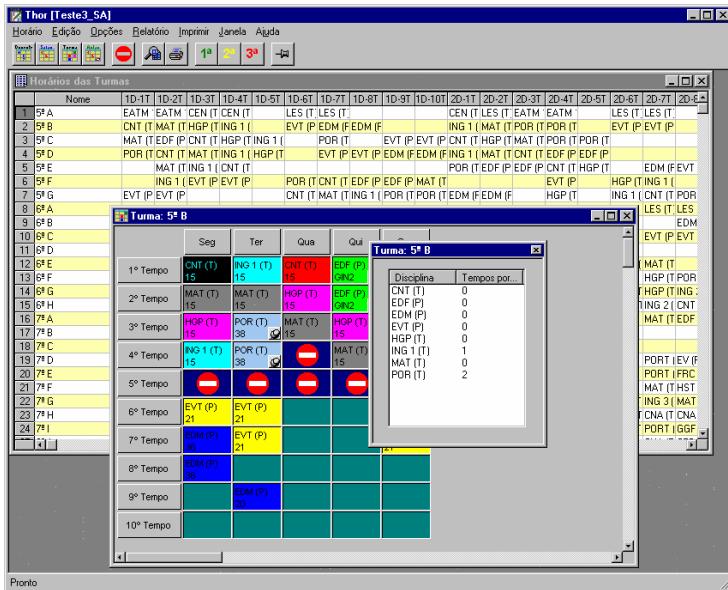


Fig. 2. Graphical user interface.

An interesting feature in this editor is that if you change the class schedules all the others are updated accordingly (teacher, room, etc.).

The Automatic scheduler contains two different and complementary algorithms:

1. An iterative algorithm based on a Fast Simulated Annealing implementation 5.
2. A heuristic constructive algorithm 4.

The Fast Simulated Annealing implementation is basically a typical Simulated Annealing algorithm 1 initiated at a lower temperature depending on the quality of the initial solution.

Also another important aspect of this particular implementation is the fast evaluation of a new solution as it is only evaluated the difference between the two solutions. It can be shown that this is most effective when a new solution only differs a small portion from the original solution 6.

There is also a deterministic algorithm for lessons that aren't scheduled. Its main objective is to schedule these lessons in any time slot available that satisfies the hard constraints.

2 Conclusion

As it is well recognized people in general are not interested in solving their optimization problems to optimality or even close to optimality. As Burke et. al. 2 stated people are more often interested in "*good enough – soon enough – cheap enough*" solutions to their problems.

We also think that good choices of specific parts of each problem are fundamental for the success of any search algorithm. As it is well known this specific problem is one with a large search space so it is very important to devise efficient techniques to search a good solution in it.

With this tool we intend to solve this problem if not to optimality at least to a very good stage with scarce resources.

References

1. Aarts, E. H. L., Van Laarhoven, P. J. M., Korst, J. H. M., "Simulated annealing", *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra (eds.), John Wiley & Sons, 1997.
2. Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S., "Hyper-Heuristics: An Emerging Direction in Modern Search Technology". In *Handbook of Meta-Heuristics*, Glover, F., Kochenberger, G., (eds.), pp. 457-474, Kluwer, 2003.
3. de Werra, D. "An introduction to timetabling". *European Journal of Operational Research Society*, v. 19, pp. 151-162, 1985.
4. Hilbert H., "High School Timetabling in Germany-Can it be done with MIP?". In *Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling*, pp. 325-327, 1997.
5. Melício, F., Caldeira, P., Rosa, A., "Solving Timetabling Problem with Simulated Annealing". Filipe, J. (ed.), Kluwer Academic Press, pp.171-178, 2000.
6. Melício, F., Caldeira, P., Rosa, A., "Two Neighbourhood Approaches to the Timetabling Problem", *PATAT 2004, Practice and Theory of Automated Timetabling*, pp. 267-282. Pittsburgh, USA, Aug. 2004.

Automated System for University Timetabling

Keith Murray and Tomáš Müller

Purdue University, West Lafayette IN 47907, USA

kmurray@purdue.edu

muller@purdue.edu

1 Introduction

Although university course timetabling is a widely studied topic, the use of automated timetabling systems is not widespread among large universities. This is particularly true in the United States, where the state of the art is typically to roll forward the last like semester's timetable and make adjustments to room assignments. University timetabling is a hard problem because of its size and the complexity of constraints needed to satisfy the demands of students and instructors. The problem is made harder yet by the need to develop a system that is easy for everyone involved in the process to use and understand, and for them to be satisfied with the results.

The system described here, and currently being implementation by Purdue University, successfully deals both with the issue of solving a large-scale problem and with addressing many of the human factors required in real applications.

The size of the problem (9,000 classes, 570 rooms, and 39,000 students with 259,000 class requests) has been made more manageable by decomposing it into a large lecture problem, consisting of centrally scheduled classes serving students in many disciplines, and multiple departmental problems. This partitioning also addresses the need to give departmental timetablers ownership of the process, which is important in a complex organization. Departmental timetablers have invaluable knowledge about what is and is not important in a solution that would be extremely difficult to incorporate into a black box solver. It is important to be able to leverage this knowledge with tools that can help sort through all of the constraints and costs to find solutions that satisfy user needs. The primary design goal was therefore to assist academic timetablers with the problem of building a good timetable, not necessarily finding a true optimal solution.

2 System Architecture

The system has been designed with a completely web-based interface (see Fig. 1) using the Enterprise Edition of Java 2 (J2EE), Hibernate, and Oracle Database.

The solver is based on an iterative forward search algorithm [3, 5]. This algorithm is similar to local search methods; however, in contrast to classical local search techniques, it operates over feasible though not necessarily complete

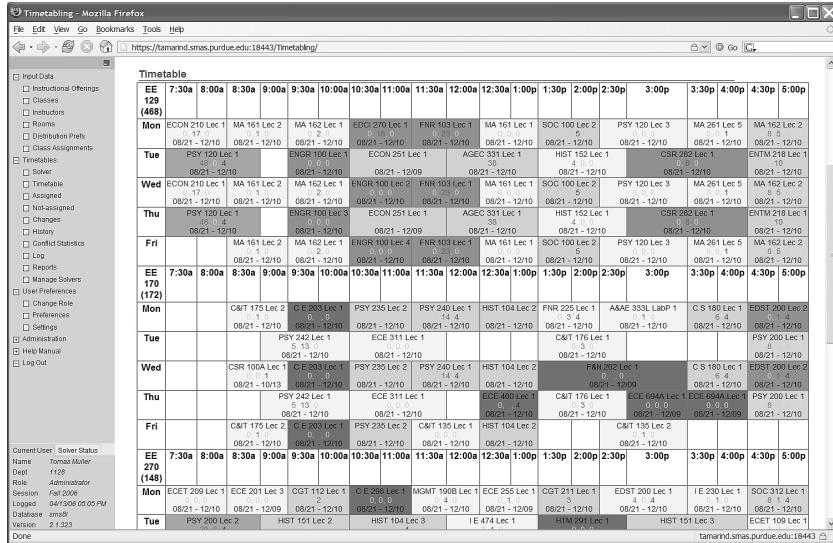


Fig. 1. Screen displaying timetable generated by solver.

solutions. In such a solution, some classes may be left unassigned. Still, all hard constraints on assigned classes must be satisfied. Such solutions are also 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 easily start, stop, or continue from any feasible solution, either complete or incomplete. Moreover, the algorithm is able to cover dynamic aspects of the minimal perturbation problem [1, 5], allowing the number of changes to the solution (perturbations) to be kept as small as possible.

3 Critical Aspects of Application

In the course of developing a system that is useable 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 the solution across all departments with classes being timetabled, ability to check and resolve inconsistencies in input data, ease of introducing changes after a solution has been generated, creating and managing constraints and other data, and dealing with incomplete demand information for classes.

3.1 Competitive Behavior

A complicating aspect of real problems in educational timetabling 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 [2].

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.

3.2 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 original design philosophy of wanting to minimize the number of changes needed to a solution [1, 5], 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 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 2 shows a list of suggestions (nearby feasible solutions) provided to the user for a given class. The user may either pick one of these alternative solutions, ask the solver to provide additional suggestions by increasing the search depth (only two changes are allowed by default), or assign the class manually by selecting one of its possible placements. In this last case, a list of conflicting classes is shown together with a list of suggestions for resolving these conflicts.

The screenshot shows a web browser window with the URL <https://tamarind.smas.purdue.edu:18443 - Suggestions - Mozilla Firefox>. The main content area is titled "Suggestions".

Current Assignment of CSR 342 Lec 1

Date:	08/21/06 - 12/10/06
Time:	MWF 7:30a - 8:20a
Room:	WTHR 200
Instructor:	Smith, John
Initial Assignment:	MWF 7:30a - 8:20a WTHR 200
Student Conflicts:	7x ECON 210 Lec 1 MW 7:30a - 8:20a EE 129 [hard] 2x AGEC 217 Lec 3 MWF 7:30a - 8:20a LILY 1105 1x PHRM 301 Peo 1 M 7:30a - 8:20a RHPH 172 [hard]
Room Locations:	4 (CL50 224, EE 129, LILY 1105, WTHR 200 , ...)
Time Locations:	10 (MWF 7:30a , MWF 8:30a, MWF 9:30a, MWF 10:30a, ...)
Room Capacity:	445

Suggestions

Score	Class	Date	Time	Room	Students
+64.3	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 12:30p	WTHR 200 → CL50 224	+29 (h+18)
+75.5	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 11:30a	WTHR 200 → EE 129	+44 (d+9,h+21)
	FNR 103 Lec 1	08/21/12/10	MWF 11:30a → MWF 12:30p	EE 129 → CL50 224	
+77.7	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 11:30a	WTHR 200 → EE 129	+47 (d+8,h-15)
	FNR 103 Lec 1	08/21/12/10	MWF 11:30a → MWF 7:30a	EE 129 → WTHR 200	
+81.9	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+74 (d+24,h+31)
	CGT 141 Lec 1	08/21/12/10	MWF 4:30p → MF 12:30p	LILY 1105 → CL50 224	
+81.9	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 4:30p	WTHR 200 → LILY 1105	+74 (d+24,h+31)
	CGT 141 Lec 1	08/21/12/10	MWF 4:30p → MW 12:30p	LILY 1105 → CL50 224	

(all 1152 possibilities up to 2 changes were considered, top 5 of 12 suggestions displayed)

Placements

Score	Class	Date	Time	Room	Students
+28.5	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 11:30a	WTHR 200 → EE 129	-4 (d+8,h-21)
	FNR 103 Lec 1	08/21/12/10	MWF 11:30a → not-assigned	EE 129 → not-assigned	
+37.1	CSR 342 Lec 1	08/21/12/10	MWF 7:30a → MWF 10:30a	WTHR 200 → CL50 224	+43 (h+14)
	COM 318 Lec 1	08/21/12/10	MWF 10:30a → not-assigned	CL50 224 → not-assigned	

Done

Fig. 2. Window displaying current assignment and suggested alternative assignments.

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.

3.3 Data Consistency

Very often, especially during an early stage of the timetabling process, the input data provided by timetablers 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 [4, 5], 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.

3.4 Data Management

A major requirement for making the system usable across campus was ease of managing data on the classes to be timetabled and using that data for other existing processes. This led to a redesign of the timetabling database from one focused on a set of classes needing time and room assignments to one that better reflected the structure of various instructional offerings, with relationships between lectures, labs, etc. This was particularly valuable for departments with many offerings of the same class. The course structure could then be used to set constraints on large numbers of related classes (see Fig. 3).

	Projected Demand	Room Limit	Time Pattern	Preferences					
				Time	Bldg	Room	Features	Distribution	Instructor
ME 200	412	412	427						
Lecture		427	3 x 50		WTHR		Computer		
Recitation		440	2 x 50			MTHW 210 PHYS 112			
Laboratory		450	1 x 50				PhotoLab		
Practice Study Observation		440							
Lec 1		427	3 x 50		WTHR		Computer		John Smith
Rec 1		220	2 x 50			MTHW 210 PHYS 112		back-to-back	Josef Novak
Lab 1		75	60	1 x 50			PhotoLab		Joe Bing
Lab 2		75	60	1 x 50			PhotoLab		
Lab 3		75	60	1 x 50			PhotoLab		
Rec 2		220	2 x 50			MTHW 210 PHYS 112		back-to-back	
Lab 4		75	60	1 x 50			PhotoLab		M E 200 Rec 2
Lab 5		75	60	1 x 50			PhotoLab		M E 200 Rec 1
Lab 6		75	60	1 x 50			PhotoLab		
Pso 1		220	0						
Pso 2		220	0						

Fig. 3. Instructional offerings contain component classes in a logical structure reflecting the relationship among these classes. Constraints may be set on individual classes or on sets of classes of the same instructional type.

3.5 Student Demand and Sectioning

The primary optimization criterion in this problem is minimizing the number of conflicts between classes that are selected by students. Other preferences are

weighted against the number of student conflicts they may cause. Since demand data is not available for all students at the time the timetable must be created (e.g., specific course selections of incoming first year students are not known at the time the fall timetable is built), it is also necessary to incorporate projected information on student course selections into the joint demand matrix for classes. This complicates the initial sectioning process and requires additional algorithms for sectioning new students consistent with the best solution that has been found.

4 Conclusions

The system demonstrated here provides a complete solution to the course timetabling problem at Purdue University. It contains an attractive, intuitive user interface along with a solver that can be used in a variety of modes, ranging from a fully automated solution to assisting with manual assignments. Currently, the system is used by the central scheduling office for the large lecture timetabling problem. Initial use by departmental timetablers will begin for the spring 2007 term, with full distribution in time for planning the fall 2007 term.

From testing done on the large lecture problem (800 classes, 50 rooms, 86,000 class requests), the solver was proved to be able to stably provide better solutions than the previous manual solutions. For fall 2005 (last semester for which a manual solution to the large lecture problem was constructed), the solver was able to provide complete feasible solutions with approximately 1.2% more satisfied class requests (i.e., about 1000 class requests), leaving fewer than 0.6% class requests violated. It was also able to satisfy more preferences on time and space. Finally, it takes approximately 10 minutes for the solver to come up with a complete high quality solution, which is a significant improvement over a week of manual work.

References

- [1] Roman Barták, Tomáš Müller, and Hana Rudová. A new approach to modeling and solving minimal perturbation problems. In *Recent Advances in Constraints*, pages 233–249. Springer Verlag LNAI 3010, 2004.
- [2] Edward L. Mooney, Ronald L. Rardin, and W.J. Parmenter. Large scale classroom scheduling. *IIE Transactions*, 28:369–378, 1996.
- [3] Tomáš Müller. *Constraint-based Timetabling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2005.
- [4] Tomáš Müller, Roman Barták, and Hana Rudová. Conflict-based statistics. In *EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics*. 2004.
- [5] H. Rudová T. Müller, R. Barták. Minimal perturbation problem in course timetabling. In Edmund Burke and Michael Trick, editors, *Practice And Theory of Automated Timetabling, Selected Revised Papers*, pages 126–146. Springer-Verlag LNCS 3616, 2005.

An integrated Framework for Distributed Timetabling

Peter Wilke

University of Erlangen-Nuernberg, Germany,
wilke@cs.fau.de

1 The Approach

We have been experimenting with software solutions for time tabling problems. This led to a requirements list for the *perfect*¹ framework:

- support of all available algorithms,
- optimal parameter settings for the algorithms,
- fast computation,
- easy to use GUI.

While the last two issues are fairly obvious the first two are worth a closer look.

- Support of *all* available algorithms requires that the algorithms have to be encapsulated and share a common interface, otherwise all other components of the framework would have to be implemented more than once.
- Setting the parameters right is the heart and soul of all optimisation algorithms. The ideal frame work would do the job just by itself or with as little user interaction as possible.

Our approach is based on the following observations:

- timetabling algorithms are optimisation algorithms,
- finding the right parameters is an optimisation problem.

Consequently we constructed our framework around a very general approach:

- the solution of a timetabling problem is found by executing an *experiment*.
- An experiment consists of several, independent *sequences* of computation steps.
- Each *step* applies an algorithm to its input and produces an output.
- Each input is a description of some data, this could be a timetable, a rooster, or a set of parameters of some other step in the sequence.
- Each output is a description of some data, this could be an optimised timetable, rooster or a set of parameters.

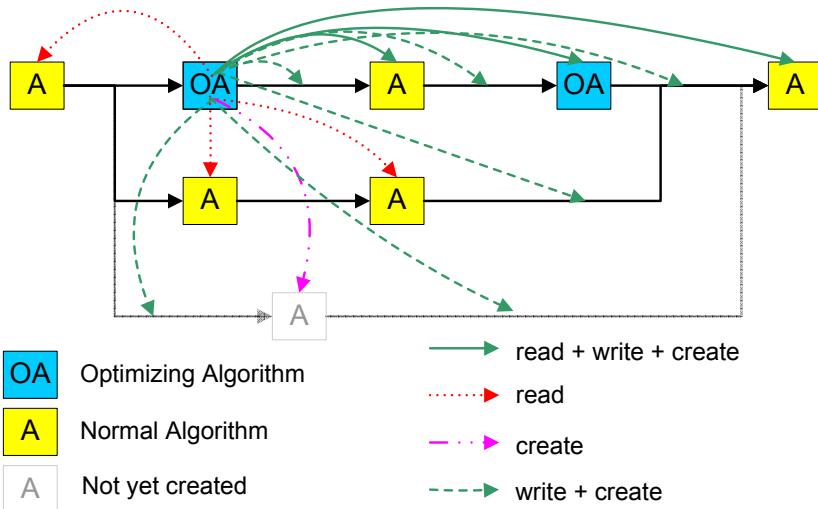


Fig. 1. Different ways an optimising algorithm can interact with the steps of his sequence.

The key idea is that the sequences can modify themselves, i.e. the output of a computational step can add new steps and branching points to the sequence (fig. 1), to avoid problems no steps can be deleted once they are created.

As a result complex control structures can be created. Very common are conditional loops (fig. 2). The optimising algorithm (OA) decides whether or not to append new steps to the sequences and puts a copy of itself as new last step in the sequence.

This mechanism allows to specify scenarios like the following:

1. generate a random timetable,
2. apply a timetable-problem-solving-algorithm using its default parameters and compute a solution,
3. stop, if the solution is good enough, continue, otherwise, and append two new steps: a parameter-optimising-algorithms and a timetable-problem-solving-algorithm to the end of the sequence,
4. apply a parameter-optimising-algorithm and compute an optimised set of parameters,
5. continue with step 3.

2 The Algorithms

Currently the following timetabling-problem-solving algorithms are available: Genetic / Evolutionary algorithms, Branch-and-Bound, Tabu Search, Simulated

¹ It hasn't escaped our attention that this is a highly subjective definition. But we do believe that most readers will share our opinion.

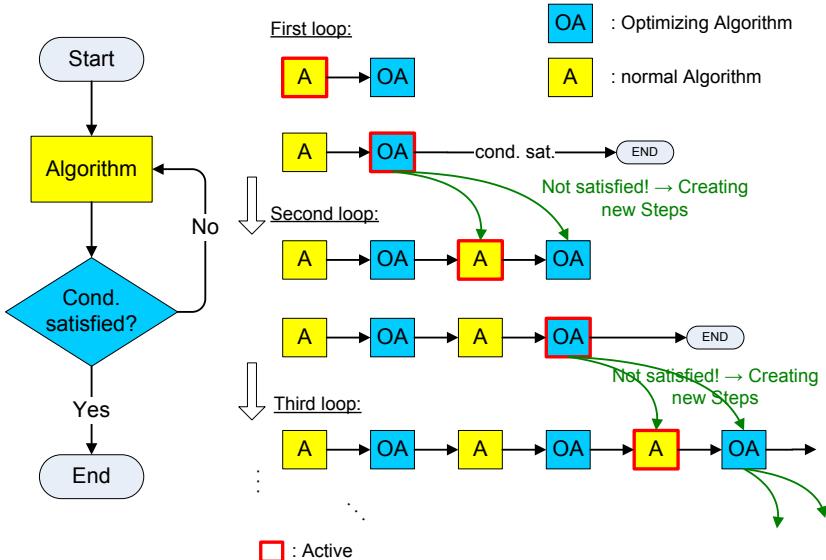


Fig. 2. Expansion of a loop in a sequence of steps.

Annealing. The following algorithms will be available in the near future: Graph Colouring, Soft Computing, Ant Colony Methods, Hybrid Methods.

The optimising algorithms are quite simple: basically they evaluate a condition and compute new parameter settings.

3 The Implementation

The software is available for Windows and Linux based systems. The software is implemented in Java 5. The core consists of an application server (JBoss), a middle-ware persistence-layer (Hibernate) and a SQL-database. Distributed computing is provided by an RMI based mechanism to distribute the computation steps on a cluster of interconnected (TCP/IP) computers.

4 The Experiments

Tabu search (TS), simulated annealing (SA) and the genetic algorithm (GA) have been applied to three real world problems. All runs were timed on a 1.7 GHz P4 PC with 512 MB RAM. Average rounded times are given.

A monthly rooster for a hospital ward with 21 nurses and 4 shifts per day was planned.

A timetable for a school with 113 teacher, 100 rooms and 43 classes was generated.

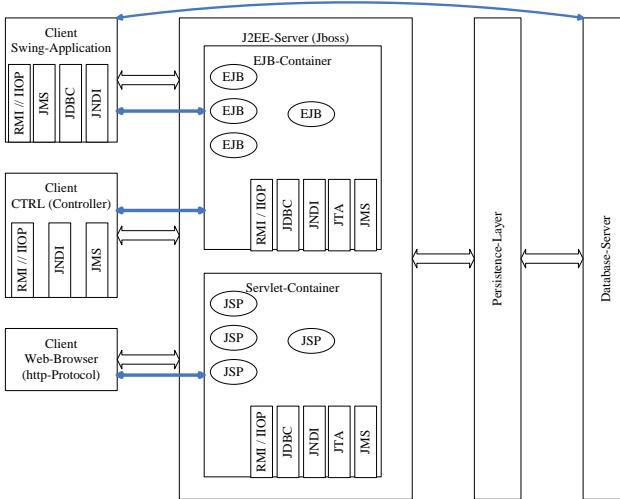


Fig. 3. The basic components of the software architecture

Our university organises a girl-and-technology week each year to attract more female students to technical subjects. In 2002 303 girls participated in 199 half-day projects. Each girl listed up to 4 first-preference projects she would like to attend, 4 substitute projects, and a list of best-friends she would like to attend the same projects.

For this problem the results are as follows:

Algorithm	time [s]	fitness
TS	300	450
GA	3500	750
SA	3500	430

5 Summary

We described a new framework to solve timetabling problems. The key features are:

- a set of ready-to-use off-the-shelf algorithms,
- experiments for automated generation of solutions,
- optimising algorithms can influence the sequence of computation steps,
- algorithm can be applied to problems or to other algorithms.

The framework has been successfully tested with several real world problems.

6 Acknowledgement

I would like to thank Matthias Gröbner, Norbert Oster, Stefan Büttcher, Adel Habassi, Hichem Essafi, Gerlinde Gagesch, Sven-Dimo Korsch, Georg Götz, Andreas Konrad and Ronny Heft for their contributions to our software.

References

1. Eric Taillard Alain Hertz and Dominique de Werra. Tabu search. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.
2. Edmund Burke and Patrick De Causmaecker, editors. *Springer Lecture Notes in Computer Science*, volume 2740. Springer-Verlag, Juli 2003.
3. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
4. Matthias Gröbner, Peter Wilke, and Stefan Büttcher. A General View on Timetabling Problems. In Burke and Causmaecker [2].
5. Bruno R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*, chapter Simulated Annealing, page 474. <http://www.brpreiss.com/books/opus5/html/page474.html>, 1998.
6. Peter J.M. van Laarhoven. *Simulated annealing*. D. Reidel Publishing Company, 1987.
7. William T. Vetterling William H. Press, Saul A. Teukolsky and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*, chapter 10.9, pages 444–447. Cambridge University Press, 1992.

Author Index

- Mieke Adriaen, 330
Samad Ahmadi, 281
Christian Artigues, 64
Hishammuddin Asmuni, 82
Masri Ayob, 336
- Ruixin Bai, 345
Mauro Bampo, 464
Amotz Bar-Noy, 351
Odile Bellenguez-Morineau, 391
Greet Vanden Berghe, 330, 426, 443
Camille Beyrouthy, 103, 359
Burak Bilgin, 123
Andreas Bortfeldt, 439
Wojciech Bożejko, 363
Dirk Briskorn, 367
John van den Broek, 141
François Bry, 496
Edmund K. Burke, 82, 103, 336, 345, 359, 370, 373, 510
Yuri Bykov, 370
- João P. Caldeira, 532
Patrick De Causmaecker, 330, 426
Tomáš Černý, 381
Peter Chan, 157
Deborah Chesnes, 391
Marco Chiarandini, 406
Tim Curtois, 376
- Abdelaziz Dammak, 384
Peter Demeester, 330
Laurens Fijn van Draat, 376, 516
Laure-Emmanuelle Drezet, 391
Alexandre R. Duarte, 394
Guillermo Durán, 398
- Michael Eley, 167
Abdelkarim Elloumi, 384
- Jacques Ferland, 2
Martin Fischer, 528
Nobutomo Fujiwara, 402
- Jonathan M. Garibaldi, 82
Luca Di Gaspero, 53, 406
Hermann Gehring, 439
Cumhur A. Gelogullari, 412
Michel Gendreau, 2, 64
Bernard Gendron, 2
Dagan Gilat, 518
Ruben Gonzalez-Rubio, 416
Dries Goossens, 420
- Peter de Haan, 423
Noureddine Hail, 2
Stefaan Haspeslagh, 426
Patrick Healy, 432
Michael Hiroux, 157
Han Hoogeveen, 437
Cor Hurkens, 141
- Shinji Imahori, 402
- Frank Jacobsen, 439
Brigitte Jaumard, 2
- Hichem Kamoun, 384
Graham Kendall, 336, 345, 443, 510
Ahamad Tajudin Khader, 234
Jeffrey H. Kingston, 181, 196
Yuuki Kiyonari, 448
Yuri Kochetov, 454
Emin Erkan Korkmaz, 123, 303
Karl-Heinz Krempels, 209, 524
Ann S. K. Kwan, 458
Raymond S. K. Kwan, 458, 473
- J. Dario Landa-Silva, 103, 359
Ariel Landau, 518

- Ronald Landman, 423
Sophie Lapierre, 2
Ignacio Laplagne, 458
Rasaratnam Logendran, 412
- Muhammad Rozi Malim, 234
Fabian Märki, 528
Philippe Mathieu, 502
Tomomi Matsui, 402, 460
Barry McCollum, 15, 82, 103, 345, 359, 373, 510
Paul McMullan, 103, 359, 373, 510
Fernando Melício, 532
Carol Meyers, 36
Wim Miserez, 443
Eiji Miyano, 448
Ryuhei Miyashiro, 402, 460
Shuichi Miyazaki, 448
Douglas Moody, 351
Tomáš Müller, 536
Keith Murray, 536
Adli Mustafa, 234
- Emilina Nano, 320
Jim Newall, 510
Diem-Hang Nguyen-Ngoc, 320
Thiago F. Noronha, 398
- Polina Obuhovskaya, 454
Jan-Kees van Ommeren, 376
James B. Orlin, 36
Ender Özcan, 123, 246, 303
- Andriy Panchenko, 209, 524
Andrew J. Parkes, 103, 359
Mikhail Paschenko, 454
Radomír Perzina, 264
Gilles Pesant, 2
Patrick Pleass, 464
Gerhard Post, 376, 423, 516
- Rong Qu, 373
Malek Rahoual, 467
- David Ranson, 281
Prapa Rattadilok, 473
Pascal Rebreyend, 478
Amnon Ribak, 518
Celso C. Ribeiro, 394, 398, 481
Franca Rinaldi, 484
Agostinho Rosa, 532
Louis-Martin Rousseau, 64
Hana Rudová, 381
Henri Ruizenaar, 423
- Rachid Saad, 467
Andrea Schaerf, 53, 406, 487
Jan Schreuder, 492
Paolo Serafini, 484
Yossi Shiloach, 518
Patrick Soriano, 2
Sebastián Souyris, 398
Frits C. R. Spieksma, 420
Stephanie Spranger, 496
- Pascal Tellier, 293
- Özgür Ülker, 303
Sebastián Urrutia, 394, 481
- Bart Veltman, 516
Marie-Hélène Verrons, 502
Manfred Vogel, 528
- Mark Wallace, 464
Segev Wasserkrug, 518
Georges Weil, 157
Andrés Weintraub, 398
Christine A. White, 320
George M. White, 293, 320
Peter Wilke, 542
Mieczysław Wodecki, 363
Gerhard Woeginger, 141
Mike B. Wright, 506
- Andrea Zampieri, 487